

国际电气工程先进技术译丛

Springer

自主移动机器人行为建模与控制

Modelling and Controlling of Behaviour for Autonomous Mobile Robots

（德） Hendrik Skubch 著

连晓峰 等译





机械工业出版社
CHINA MACHINE PRESS

国际电气工程先进技术译丛

自主移动机器人行为建模与控制

（德）Hendrik Skubch　著

连晓峰　等译



机 械 工 业 出 版 社

多智能体协调控制是机器人和人工智能领域的研究热点。本书主要涉及多机器人（多智能体）的协调控制问题，提出了一种交互式协作智能体语言 ALICA（交互式协作智能体语言），详细描述了 ALICA 的语法、语义、冲突检测与消解、软件架构、约束问题求解等内容。最后，通过三种场景，即机器人足球、探索和搜救来评估验证所提方法的有效性。

本书可作为多机器人协调控制、人工智能和计算机科学领域的研究人员的参考书，也可作为高等院校相关专业研究生以及教师的参考用书。

图书在版编目（CIP）数据

自主移动机器人行为建模与控制/（德）司库巴赫（Skubch，H.）著；连晓峰等译．—北京：机械工业出版社，2014.5
（国际电气工程先进技术译丛）
书名原文：Modelling and controlling of behaviour for autonomous mobile robots
ISBN 978-7-111-46357-3

Ⅰ.①自…　Ⅱ.①司…②连…　Ⅲ.①移动式机器人－研究　Ⅳ.①TP242

中国版本图书馆 CIP 数据核字（2014）第 066513 号

机械工业出版社（北京市百万庄大街 22 号　邮政编码 100037）
策划编辑：顾　谦　责任编辑：顾　谦
版式设计：霍永明　责任校对：纪　敬
封面设计：马精明　责任印制：乔　宇
北京机工印刷厂印刷（三河市南杨庄国丰装订厂装订）
2014 年 6 月第 1 版第 1 次印刷
169mm×239mm・12.75 印张・238 千字
0 001—3 000 册
标准书号：ISBN 978-7-111-46357-3
定价：59.90 元

凡购本书，如有缺页、倒页、脱页，由本社发行部调换

电话服务	网络服务
社服务中心：(010)88361066	教材网：http://www.cmpedu.com
销售一部：(010)68326294	机工官网：http://www.cmpbook.com
销售二部：(010)88379649	机工官博：http://weibo.com/cmp1952
读者购书热线：(010)88379203	**封面无防伪标均为盗版**

译 者 序

本书首先介绍了自主移动机器人行为建模与控制的研究意义和所涉及的相关研究，并阐述了智能体、多智能体、约束规划等理论基础，在简单描述一些相关理论和技术的基础上，提出了两种方法，分别是命题交互式协作智能体语言和通用交互式协作智能体语言。详细描述了 ALICA（交互式协作智能体语言）的语法、语义、冲突检测与消解、软件架构、约束问题求解等内容。最后，通过三种场景，即机器人足球、探索和搜救来评估验证所提方法的有效性。

本书是涉及多机器人协调控制方面的专著，机器人研究领域的成果很多，相关的理论与方法也很多，本书的创新点在于提出一种交互式协作智能体语言 ALICA 来解决在动态环境下，机器人实时反应并在通信不可靠条件下进行冲突检测与消解，通过约束满足与约束规划等理论来实现多机器人之间的协调控制。这对于从事该领域工作的工程人员和高年级学生有着重要的参考作用。

作者 Hendrik Skubch 曾是卡塞尔大学的助理研究员，现在日本 Square Enix 公司工作，长期从事多机器人协调控制方面的研究，出版过多部相关著作。

本书主要由连晓峰翻译、审校、整理，并对原书中的错误进行了注释，王佩荣、潘媛、金成学、李东红、贾琦、张子康、王宇龙、郭柯、王伟和张云等人参与了部分内容的翻译。

本书可作为多机器人协调控制、人工智能和计算机科学领域的研究人员的参考书，也可作为高等院校相关专业研究生以及教师的参考用书。

限于译者的经验和水平，书中难免存在缺点和错误，敬请广大读者批评指正。

原书前言

随着研究的不断深入与机器人能力的不断提升，机器人系统也在越来越多的情况下具有切实可行的解决方案。这些成果尤其应用在多机器人系统中，将单机器人的能力与功能相结合。未来自主移动机器人的鲁棒性、效率以及自适应性等关键特性都在复杂性和动态环境下得到充分应用。因此，不同建模和推理模式所引发的问题都可通过预期行为的描述来组合实现，并达到上述特性。本书提出了一种实现自主移动机器人团队行为的建模与执行的综合解决方案。所提出的框架ALICA（交互式协作智能体语言）与建模技术相结合，以一种综合方式推导出不同模式。利用有限状态机的层次结构来构建机器人团队的行为，由此可表示其时间关系和因果关系。利用效用函数来衡量各自的不同选择，并对不同智能体分配不同任务。最后，将约束满足和优化问题相集成，可用于以一种简洁且具有良好理论基础的方式来指定复杂的协作行为。该系统是针对要求机器人必须快速反应的高度动态环境，通信不可靠且单个机器人随时可能损坏的环境。在该环境下，要求智能体在面临不断变化的情况下必须及时反应，而不是采用预先设定的协议。由此，ALICA 智能体可局部决策并在建立通信之前相应动作。由不一致的决策和信任所产生的冲突可以可靠地检测和解决，这是由于 ALICA 完全分布式工作，不会出现单个故障。

从建模角度来看，ALICA 作为一种现代编程语言可对机器人团队进行全局建模。通过层次结构和程序组件的抽象可提高主题透明性和可重用性的复杂度。状态机、效用函数和非线性连续约束满足和优化问题相结合为描述机器人团队行为提供了一种为目前技术全新的方法。

执行层采用一种新的时间算法来解决综合约束问题，由此使得在动态环境中随时间跟踪解决方案，协调团队间的解决方案，并利用团队内部的分布式计算能力。

将该方法应用于机器人足球来进行评估，相对于非可靠通信，更关注于反应性和鲁棒性。在外星探索领域中，勾勒出一些先进技术来描述机器人动态编队。最后，对 ALICA 的可扩展性进行研究，并利用一个常用搜救场景与目前最先进的方法进行比较。

原 书 致 谢

在此衷心感谢我的博士导师 Kurt Geihs，他在卡塞尔大学分布式系统团队创造了美好的氛围，思想自由和热情友好的精神使这里独一无二。另外，我还要感谢我的副导师，Alexander Kleiner，他付出了大量心血，让我受益匪浅。

对本书贡献最大的是 Roland Reichle 和 Philipp Baer，他们在我到卡塞尔大学工作之前，已成功地完成组建机器人足球中型组比赛队伍的工作。与此同时，我非常珍惜与各位同事共事的时光，Thomas Weise 经常为我提供大量建议，与我分享其独特观点，Michael Wagner 或许是最好的同事，Steffen Bleul 总能让我们脚踏实地工作，Michael Zapf 常常会进行富有成效的讨论。还有 Diana Comes 和 Christoph Evers，他们也为团队的友好氛围做出了大量贡献。同时，非常感谢 Thomas Kleppe、Iris Robbach 和 Heidemarie Bleckwenn 等工作人员，他们时常为我们提供无私的帮助。

Carpe Noctem 目前由 Dominik Kirchner、Daniel Saur 和 Andreas Witsch 领导，他们为机器人的实现提出了引人入胜的想法。与你们共度的时光太美妙了！非常感谢 Carpe Noctem 团队，不管是过去还是现有的成员，正是因为他们的努力工作才有团队现在的成绩：Till Amma、Kai Baumgart、Jewgeni Beigub、Fridolin Gawora、Tareq Haque、Janosch Henze、Timo Heumuller、Kai Liebscher、Claas Luhring、Stefan Jakob、Stefan Niemczyk、Stephan Opfer、Stefan Triller、Andreas Scharf、Jens Schreiber、Martin Segatz、Florian Seute、Daniel Walden 和 Martin Wetzel。我永远不会忘记我们共同工作、进行机器人、实验或仅仅是一起闲逛的美好时光，哪怕是机器人坏掉。

最后，感谢我的父母，是他们的无私支持让我取得现在的成就，另外也非常感谢 Caroline Wagenaar 的支持与耐心讨论，这不断激励着我继续前行。

Hendrik Skubch

目　录

第1部分　预备知识

第2部分　命题式ALICA

第 4 部分　评　　估

第1部分　预备知识

第1章　引　　言

1.1　研究目的

自从 Shakey 机器人通过 STRIPS（斯坦福研究所的问题求解器）[45]控制其行为在斯坦福大学里自主漫游开始，个体机器人就已成为一个非常活跃的研究领域[139]。尽管自此以后，在机器人、人工智能和机器学习领域取得了大量的研究成果，但直到近年来机器人才开始出现在人们的日常生活中。如今，机器人已作为社会辅助技术成功应用于医院[9]的外科手术工具[165]，并预计在不远的将来可用于家庭护理[162]。

然而，随着机器人与现代社会和经济的深入结合，将不断要求机器人在现代化日常生活的动态环境中需具有智能性和自适应性行为能力。除了日常生活之外，机器人的另一个主要目标是为人类无法到达或具有高度危险性的区域提供一种设计巧妙的机器。这些领域包括灾难性事件的搜救与救援、太空探索或未来小行星开采[68]。而且，这些机器还可完成过于重复性的任务。

在上述应用场景中，潜在的异构机器人团队具有较大优势。团队机器人比个体机器人具有更高的鲁棒性，并可在同一时间内搜索更大的区域。此外，对于一个已处于工作状态的机器人，集成一个新机器人通常要比集成一个新的特殊功能部件更容易。然而，与个体机器人操作相比，让团队机器人以高效、鲁棒的方式进行工作要面临更大的挑战。整个团队应相互协调，对动态环境作出协调一致的反应，并对丧失行为能力的团队成员进行补偿。所有这些问题都需要以一种自适应分布式方式实时处理，这是因为采用任何一种集中控制机制都将失去团队机器人的最大优势，即对于发生故障部件的鲁棒性。

在上述情况下，创建一个可实现期望目标的解决方案或规划是一项艰巨任务，无论该解决方案是否由规划算法生成，或由相关领域专家建模，或在规划过程中交互生成。因此，具有期望行为的上述解决方案的底层执行至关重要，即使在感知噪声较大、团队成员之间不可靠通信以及条件动态变化的困难情况下。

在更高层解决方案与团队表现行为的相互关系方面，解决方案的编程语言是一项重要因素。编程语言，或对编程语言的基本表征，必须被所有参与方理解，包括系统开发人员、规划算法或执行层。因此，编程语言应具有规范的语义，从而可验证个体机器人均遵循一个共同理解。另外，编程语言必须具有足够的表达能力来描述团队机器人所处的各种场景。

在搜索和救援场景中，团队机器人必须应对动态变化的情况，包括可能存在环境急剧变化、个体机器人失效以及通信不可靠等情况。在这些情况下，执行层的作用尤为重要。针对每种新情况，组件评估包括是否继续某一项行为动作，或应立即修复，或必须完全中止。然后，选择适当的修复机制，并触发更高层的元素，如规划算法（如果需要的话）。根据具体情况，或许不需要通信的迅速反应，在其他情况下，则需要团队成员之间进行通信来解决冲突。除此之外，正是该元素使得期望行为的符号描述与底层执行器相关联。换句话说，即将物理行为的动作符号具体化。这样，执行层就可支持高层规划和学习的认知能力。

最后，编程语言应能将具体部分的复杂性进行抽象，如在某一时刻，系统设计者只需关注具体问题。这就要求编程语言应支持关注点分离和组件的可重用性。

1.2　问题描述

本书的目的是为团队协作自主机器人的行为建模和控制提供一个综合解决方案。该方案包括两大部分：一是描述机器人团队行为的编程语言；二是将编程语言中的具体元素看作程序并有效和鲁棒地执行相应执行层。

为便于后续编程工具的支持，以及允许将来嵌入到其他编程语言或将其他编程语言移植进来，建模语言必须具有清晰规范的语义。根据所处的领域和场景，编程语言元素（如程序）将由开发人员编写或通过规划或学习算法自动生成。

与将在第3章中介绍的目前已有的许多编程语言和框架相比，该编程语言框架可支持全局建模，使得设计者只关注于较高抽象层次上的协作问题，而无需处理相互作用的多个具体程序。同时，该编程框架还可允许目标任务内部结构之间的精细建模。

如前所述，本书所提出的编程语言必须以一种简洁、明确的方式来实现复杂行为的规范化。由于具体行为总是与特定领域的机器人相关（如位置），因此该编程语言必须能够描述这些机器人的相关特性，同时还需保留其与领域无关的本质。在此，将一类约束满足和优化问题集成到该编程语言中，并对执行层配置一个相应的求解器。

本书主要考虑需要快速反应并推理的动态场景。更具体而言，即执行平台必

须可应对一个具有下列情况的动态环境：

- 状态连续实时变化，如在推理时。
- 传感器存在噪声，执行层需具有一定的抗扰性。
- 环境部分可见，机器人无法在长时间内有效预测其行为。
- 通信不可靠。在此假设概率大于0时才可通信，时常发生丢包和延迟。
- 个体机器人随时都可失控或丧失部分功能。只要有可能，团队机器人性能应适度降低。这不包括采用任何核心部件。

在这些特性之间存在某种程度的内部关联，这需要在本书的工作中进行平衡。首先，在处理动态环境下感知噪声时，对噪声的反应性和抗扰性是两个相互冲突的目标。同理，在高动态环境下，通信也不同步，因此所接收的消息总是针对之前状态的。尤其是，如果智能体时钟不能严格同步，随着环境的突然变化，信息包的延迟也会相应增大。一般来说，更注重对噪声的反应性，而不是抗扰性，然而本书中的编程框架可将任何一种感知处理方法和噪声处理聚类方法相结合。其次，智能体失效与数据包丢失没有区别，除非智能体动作可通过感知数据来观测。最后，反应行为和协作行为也是两个相互冲突的目标。如果进行决策时需要通信，则无法实现较高的反应行为。另一方面，通信或许是目前唯一可行的实现合作的方式。因此，需要一种自适应性方法，在没有前期通信的情况下进行快速决策，如果有必要，应切换到具有高度一致性协议的低层次反应决策。

由于与多机器人系统相关的研究领域有许多，本书中不考虑以下研究内容：

传感器融合——对于异构机器人团队中的传感器融合，Reichle[13]进行了详细讨论。在所有的场景中，均假设已具有合适的传感器融合算法，由此在该基础上进行决策。

协同系统——在协同系统中，每个个体机器人都具有潜在的不同目标，这些不同目标暂时保持一致并保证形成可实现共同目标的联合体。在此，假设所有机器人都具有相同的全局目标，并在整个生命周期内构成一个团队。

规划——规划算法可生成全部或部分有序行为序列，以实现特定目标。在此并不制定或集成规划算法，然而可提供一种易于描述规划算法的编程语言。

1.3 应用场景

接下来，简单介绍用于开发和评估本书研究成果的应用场景。值得注意的是，本书方法并不局限于这些领域，并可非常方便地应用到其他领域。在本书方法中重点强调的是特定领域机器人与普通机器人之间的不同。因此，关于领域的描述是易于更换的。

1.3.1 机器人足球

RoboCup 是在机器人、人工智能以及相关领域许多国家深入研究并努力发展的项目。其核心是不同国家联赛的一个年度竞赛，并具有一个愿景：

“到 21 世纪中叶，按照 FIFA 的官方规则，一支完全由自主人形机器人组成的足球运动员在与当时世界杯冠军的足球对抗赛中获胜” RoboCup 网站[137]。

出于这个美好愿望，RoboCup 也致力于一个主要的研究内容，即机器人足球。自从 1996 年开始首次比赛以来，RoboCup 目前已得到广泛普及并扩展了许多其他领域，如搜救机器人和家庭服务环境机器人，然而简单并具有巨大挑战的足球比赛仍是其主要关注的领域。相对于其他研究领域，足球比赛领域具有很多优势：首先，足球是一种众所皆知的体育项目，因此无需过多解释即可引起普通大众的关注；另外，还是一个以对抗方式对不同方法进行评估的测试平台；而且在足球比赛中，每个个体运动员的能力技巧与团队配合同样重要。因此，需要具备个人能力以及团队配合能力，才能组成一支优秀的机器人足球队。最后，足球是一项快节奏的项目，需要高度的反应能力和速度。这种需求使得机器人足球不同于典型的实验室配置，必须满足参加机器人足球所需要的基本要求。

本书中所涉及 RoboCup 领域，重点关注一种特定的 RoboCup 联赛，即中型组比赛（MSL）。在 MSL 中，5 个机器人组成的队伍要在一块 18m × 12m 的场地上相互对抗。该比赛包括上下两个半场，每个半场进行 15min。采用标准的 FIFA 足球进行比赛。在比赛中，每个机器人完全自主行动，严格禁止人为操作。目前，参赛机器人最大 80cm 高，不超过 40kg 重。大多数机器人采用全驱运动，以及一个独立的踢球装置进行长距离传球。机器人之间通过无线局域网进行通信。比赛是高度动态的，机器人可达到 5m/s 的速度，带球运动速度可达到约 12m/s。鉴于在理想状态下，机器人配置的典型传感器感知范围不大于 10m，通常小于 6m，对于机器人性能而言，反应速度尤为关键。除此以外，机器人还需协调一致地行动和反应。在本书的研究工作中，在保证机器人团队相互协调以形成一致行动的情况下，尽可能地保持快速反应时间。

1.3.2 探索

在自主移动机器人应用中，探索任务是一个非常有益的应用场景。执行探索任务时的环境通常是难以到达第一现场或对人类具有危险性的地方。例如，对于外星探索任务，就同时具有上述两个特点。而且，天体之间的超远距离也难以遥控机器人，甚至根本无法实现。因此，关键在于要赋予探索机器人一定程度的自治性。

IMPERA 研究项目就是针对上述场景开展研究的，该项目是由 Kassel 大学的

DFKI[㊀]机器人创新中心联合分布式系统研究团队，并与DLR[㊁]合作共同完成的。基于团队机器人要比单个机器人更具有鲁棒性和容错性，并且任务完成（如探索任务）要更加快速的思想，IMPERA主要针对外星环境下机器人的协调合作进行深入研究。

尽管在火星或月球环境下，机器人的动态性远不及足球比赛的环境，但还需考虑1.2节中讨论的其他特性。然而，团队机器人本身可通过与外界环境的交互在一定程度上降低动态性。

本书所提方法仅作为IMPERA在外星探索任务环境下协调控制机器人的基本方法。尤其值得注意的是，IMPERA利用先进规划算法对本书方法进行扩展。

1.3.3 搜救

在搜救场景下，机器人可完成多种潜在任务。例如，机器人可作为救援人员在大范围区域内搜索幸存者、提供通信网络、清除废墟、进入摇摇欲坠的建筑物。在本书中，根据RoboCup救援仿真比赛，重点研究一个简单场景。在该场景中，智能体将在城市中的多个着火点进行灭火任务。在此采用Kleiner等人[86]提出的RMAS-BENCH来模拟仿真该基本问题。在搜救领域，需要比探索或足球比赛领域更多的智能体进行协调合作。

1.4 研究方法

如上所述，本书所提出的解决方案包括一种编程语言和相应的执行层。为抽象化执行过程中行为描述和具体机器人之间的关系，采用行为描述中机器人与任务之间的两个映射关系。根据具体执行器和感知器、智能体或机器人所具有的个体能力与相应的角色相对应。这些角色又决定着该智能体所能完成的任务以及执行程度。这种映射关系完全按照Wooldridge等人[182]所提出的思想。由此，团队机器人行为与运行时的具体机器人相互独立，而且团队机器人也可独立指定后续应解决的问题。

团队行为程序是一种多个有限状态机的层次结构。状态机的每个状态中都包含多个子程序，这意味着可由继承该状态的所有智能体执行。每一层中的条件和效用函数决定哪个智能体执行哪种状态机。每个智能体对这些条件和函数进行局部评估，以实现高度反应行为。该层次结构隐藏了每层中较低层次的复杂性。

㊀ 德国人工智能研究中心。

㊁ 德国宇航中心。

通过冲突检测和冲突消解机制，对基本编程语言进行扩展，即选择一个领导者来暂时切换决策协议，从而解决检测到的冲突。然后，相对于冲突时的决策，该领导者只对团队其他成员发布命令。冲突解决后，团队又切换回其初始状态，具有更多的动态决策模式。

最后，利用约束来描述机器人应具有的目标值。在动态约束优化问题中的这种结果由团队成员实时求解。在此对该任务中的非线性连续约束优化问题提供一种有效的任意求解器。随着环境以及相应的优化问题的变化，该求解器可跟踪解决方案。而且在解决难题时，团队成员相互合作，因此可比单个智能体更快地找到解决方案。最后，求解算法使得团队合作解决，从而得到一致结果。执行层管理活动约束集合，使得任何部分都按照变量值进行排列。作为声明描述的非合作约束，将其添加到编程语言中是非常有利的。

- 极大地简化建模任务，并减少所需的不可分行为的个数。
- 只要约束之间的数据关系清楚，易于对约束进行扩展和联合。例如，可直接构建两个约束满足问题之间的交集。
- 与专用程序相比，无需处理太多的特殊情况。
- 通过规划和学习算法，数学描述与具体实现的直接对应可生成约束。
- 最重要的是，约束提供一种以数值形式来进行符号行为描述的方式，这可传递到较低层次单元，如电动机控制器。

当然，所提方法也有其缺点。一个通用求解器在效率方面总是超出具体问题解决方案的执行。因此，设计一个专用程序来处理特定任务肯定会比解决同样问题基于约束的解决方案更加有效。

1.5 创新与贡献

本书的主要贡献在于为团队机器人的行为描述提供一种全面的解决方案。在此介绍的范例组合为多机器人系统的合作行为提供一种全新方法。称为 ALICA（交互式协作智能体语言）的方法包含功能描述、角色分配、类似于层次状态机的任务描述、基于效用函数的任务分配、通过同步转换的显式协调、广播计算方式的隐式协调以及通过切换正在采用的决策协议来进行冲突检测和冲突消解。最后，与非线性约束满足问题相结合，以允许特定域实体内的推理（如关节位置和配置）。在此，将该全面综合的组合方式作为本书的最大贡献。

自从 2009 年，该方法已成功应用于参加 RoboCup MSL 比赛的机器人队伍 Carpe Noctem 中，并且现已应用于 IMPERA 研究项目。在基于 BSD 的开放源代码

协议下，可提供源代码[1]。

在理论方面，将新方法引入分层任务分配中，根据任务结构的不同需求，可产生两种不同的分配机制。而且提出一种任意约束优化求解器，将现有的用于非线性连续满足问题的先进方法进行扩展。值得注意的是，该方法具有随时间跟踪解决方案的能力，并提供团队内部的合作解决方案，以及在保证单个机器人反应能力下的分布式问题求解能力。

1.6 结构安排

本书分为4个部分。第1部分中的其余章节主要包括：第2章介绍本书工作的基础知识，接下来在第3章中介绍与本书所提方法相关的研究工作。

第2部分主要详细介绍ALICA的命题式核心语言。在第4章介绍ALICA基本语法之后，第5章逐一介绍相关语义。在该部分中重点阐述任务分配和基于规则的执行，这些将构成后续章节的基础。

第6章中通过增加冲突检测和冲突消解对基本语义进行扩展。推导了域无关的冲突检测机制，这可对团队合作中的持续冲突进行可靠检测。之后，提出了一种基于局部领导者选举的冲突消解方法。在第2部分的结尾，即第7章中，详细介绍了关键的软件架构和具体实现。

第3部分，通过考虑变量和约束条件对命题式语言进行扩展。相应地，在第8章中介绍了语法和操作语义，以适用于额外的编程语言元素。接着，在第9章中推导并详细讨论了一种适当的约束求解算法。本章的其余部分解释了如何随时间跟踪解决方案，以及如何实现非命题式情况下的合作。

第4部分重点关注所提解决方案的评估和讨论。第10章中给出了全面的评估结果。利用机器人足球、探索和搜救等场景，着重讨论了不同场景下，网络较差条件下的鲁棒性，噪声条件下的鲁棒性，建模能力，可扩展性和适用性。在第11章中进行总结，并展望今后的研究工作。

1.7 惯例与约定

为简单起见，在此假设本书中的符号约定如下：

- 有限集的减法运算用 - 表示：

$$A - B \overset{\text{def}}{=} \{a \mid a \in A \land a \notin B\}$$

---

[1] http://ros.org/wiki/cn-alica-ros-pkg。

- 除另有特殊说明，公式中的自由变量均统一量化。
- 在一阶方程中采用如下缩写：

$$(\forall x \in S)\phi \overset{\text{def}}{=} (\forall x)x \in S \rightarrow \phi$$

$$(\exists x \in S)\phi \overset{\text{def}}{=} (\exists x)x \in S \wedge \phi$$

- 在通常意义下，用 img(f) 表示图像函数 f。
- 2^S表示集合 S 的幂集。
- 在特意提及术语或公式中的自由变量时，用 vars(p) 表示术语或公式 p 中的自由变量集。

第 2 章　基 础 理 论

本章简要介绍本书工作的主要基础理论：2.1 节介绍了智能体的概念；接下来的 2.2 节介绍了不同多智能体系统的分类；2.3 节主要阐述了文献中团队合作的主要理论；最后，2.4 节对约束规划进行了概述。

2.1　智能体

智能体概念已广泛应用于各种配置。研究人员将智能体作为一种软件技术实体（如参考文献［115］中所述）、一种模拟生态系统行为的方法（如参考文献［61］中所述）以及区分环境和内部的人工智能概念进行了深入研究。

“智能体是一种通过传感器感知其周围环境并通过执行器作用于环境的任何事物。” Russell 和 Norvig[139,第32页]。

这或许是文献中最通用的智能体定义。尽管各个模块中具体步骤的名称和个数或许不同，但一个智能体通常都是按图 2.1 所示的循环流程来实现的。

首先对传感器输入数据进行处理，所得结果用于可产生行为决策的推理步骤。反过来，执行该行为又将改变外界环境，由此导致不同输入。根据如何实现上述步骤，采用何种内部模型以及智能体适用于何种环境，具有多种不同的智能体分类形式。Russell 和 Norvig[139]对此进行了详细、全面的综述。在此，仅讨论其中三种不同的智能体模型，即理性智能体、BDI（信任、期望和意图）智能体和推理智能体。

理性智能体　最大限度地发挥其预期的性能指标。即以一种决策理论最优方式进行行为动作，并提供有关回报和概率发生的正确信息。

BDI 智能体　基于 Bratman[12]所提出的 BDI 模型。BDI 着重于一种具有特殊理念的实际推理方法。信任表示智能体对于环境（包括其本身和其他智能体）的信息状态；期望表示智能体想要完成或实现的目标与状况；意图表示智能体会选择何种行为的慎思状态。这种实用方法已取得很大成功，并产生基于 BDI 模型的多种编程语言。

推理智能体　按照推理是以某种逻辑形式进行演绎推理这一人工智能方面的经典观点。该领域的研究主要集中于如何对知识进行表示以适用于功能强大的推理技术。该领域的基础研究工作应追溯到 Mc Carthy 和 Hayes[100]的相关研究。推理智能体中两种最具影响力的演算方法是情境演算[134]和流演算[72,166]。

尽管 ALICA 的灵感来源于 BDI 模型，并可看作一种 BDI 语言，但本书的工作与上述三种智能体模型均兼容。在随后的内容中，“智能体”与“机器人”可相互替换。

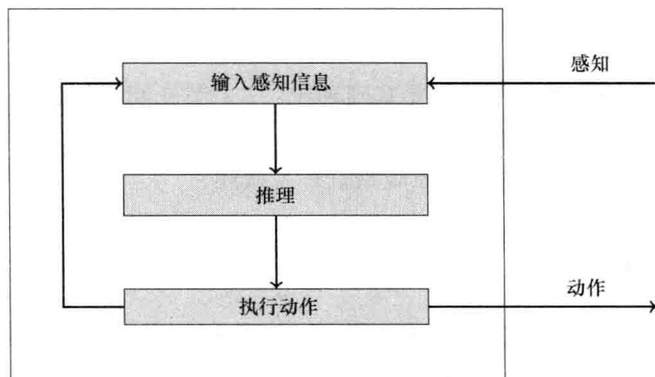


图 2.1　智能体执行循环流程

2.2　多智能体系统

根据智能体的概念，目前包含多个智能体的系统仍是研究热点。这种系统具有多种分类方法，如基于单个智能体的能力、基于所用的组织隐喻以及根据智能体是同构还是异构。Wooldridge[181] 对于多智能体系统领域进行了全面、详细的综述。其中主要是从多智能体合作方面进行阐述。一个多智能体系统可分类如下：

合作　意味着所有智能体都试图达到同一个全局目标，因此将以任何有利于实现该目标的方式相互合作。

协作　智能体无需实现同一目标，但各个子目标相互兼容。因此，每当单个子目标一致时，这些智能体就形成联盟。与合作系统相比，协作系统的组织结构将更加动态，这是由于团队成员可在系统内不断组合和解散。

中立　智能体具有各自不同的目标且相互无关，如果发生任何形式的合作都纯属偶然。

对抗　智能体具有相互直接竞争的目标，即某个智能体达到其目标后，意味着其余智能体将不能再达到该目标。

一般而言，上述分类并不严格，实际系统中可以具有各个分类的特征或特性。如机器人足球中，每个队伍都组成一个完全合作系统，且团队中的每个成员完全利他。当然，在进行比赛时，两个队伍所构成的多智能体系统是相互对抗的。

在本书的研究工作中，仅考虑完全合作的系统。当所研究的场景中具有比赛对手队伍时，并不关心分析每个足球机器人的整体行为，而只是对整个队伍的行为进行建模。换句话说，由于无法控制对手队伍的行为，因此只是将其看作外部环境的一部分。

由大量典型同构机器人组成，且每个机器人的感知与通信能力相对简单的多智能体系统是群体智能和群体机器人领域的研究热点。基于群体的方法受生物系统启发（如参考文献［119］中所述）。另一方面，将一组机器人建模为一个团队时，假设个体机器人已足以进行相对复杂的推理。在2.3节中，将介绍有关团队机器人合作的主要理论。

除了群体和团队外，还有许多其他不同的方法来组成多智能体系统。Horling和Lesser[73]针对不同典型系统进行了全面概述：

层次结构　在层次结构中，较高层的智能体要比低层智能体更具有全局观。通常，智能体只与通过树形结构直接相连的其他智能体进行通信。数据是自下而上的通信，而控制指令则是自上而下。层次结构相对简单，且与层次目标或任务树之间的关系清晰，但该结构相对固定且适应性差，具有单点故障的特点。

合弄结构（Holarchy）　合弄结构是一种类似于层次结构的自然启发式结构，其中，除某种部件外，在每个层次中构成一个群体，而这些部件又是较低层次中抽象部分的部件群体。在最底层，多个智能体构成一组。因此，合弄结构可看作不严格的层次结构，允许单个智能体之间具有更多通信以及更多的自治。

联盟　联盟是基于上述的协作智能体思想。在此，为实现共同目标，智能体构成相对短暂的平面组。

团队　团队是由为实现共同目标而共同工作的合作智能体组成的。相对于联盟，这是一个完全合作系统。

集合　集合是相对稳定的扁平型结构智能体。与团队相比，集合的组成并没有一个统一目标，而是为使各自能力互补相结合。

社会　智能体社会是一个相对稳定的开放系统，如电子市场。智能体具有各自不同目标和异构能力，并通过各种途径交互。社会具有一系列个体所必须遵循的约束条件，通常称为社会法律或规范。

联邦　在智能体联邦中，潜在的复杂智能体组由组中某个杰出成员来代表。该成员负责与其他组的代表进行相互通信和交互。

市场　相对于社会，市场中的整个交互过程经过商业交易后建模，如商品买卖、投标、提供服务等。这些交易通常都是具有竞争性的，即各自目标相互冲突。

矩阵　基于矩阵的智能体组织形式可对权限进行多维定义。单个智能体必须具有足够的自治权以应对可能由不同权限引起的潜在的局部冲突。

复合 最后一种形式基本上是将不同用途的不同组织结构相结合，如数据流、控制、探索等。

由于本书主要关注具有共同目标的小规模机器人组，因此所提方法主要分析基于团队的多智能体系统特性。

2.3 团队合作

在文献中已深入分析了智能体之间的团队合作。目前，利用智能体语言进行团队合作建模的大多数方法都是基于以下两种基础理论中的一种。

联合意图理论

联合意图框架[28,97]是基于 BDI 逻辑的理论框架。该框架关注于团队的联合心理状态，称为联合意图。如果在特定心理状态下所有团队成员共同致力于执行一个动作，那么该团队就会共同实现一个团队行为。

为实现一个共同目标，团队成员必须建立一个合适的共同信念以及单个承诺。尽管联合意图理论没有强制要求根据观测来建立行为共同信念的通信以及相关技术（参见参考文献［75］），但目前而言，通信仍是获得共同承诺的唯一可行方法。联合意图理论中一个非常关键的方面是获得有关团队行为终止的共同信念。这将有助于保证团队保持行为状态更新。通过强制智能体提交联合意向或有关故障或提前终止的团队信息来实现上述行为。联合意图和联合提交提供了一种基本框架来推理有关合作以及监视和保持团队行为的监控。然而，高层团队目标的单个联合意图不足以对团队行为详细建模，以保证一致的合作。

共享计划理论

与联合意图相比，共享计划理论[63,64]对意图采用层次结构，由此克服复杂团队任务中单个联合意图的缺点。共享计划理论并不是基于联合心理状态，而是基于称为打算的意图状态，这与一个智能体完成一项动作的通常意图非常相似。然而，单个智能体的打算是直接面向合作者行为或团队联合行为。“打算”是通过一组指导单个智能体行为（包括通信）的公理来定义的，使得便于团队合作者、子团队或团队完成所分配的任务。

对于小组行为，共享计划具体指定了关于如何进行行为和子行为的信念[63,64]。形式化模型可获得单个和群体行为性能的意图和提议。合作计划由共同信念、（部分）配方、单个执行动作的意图、合作者在其子行为中成功的单个打算以及子行为的单个或合作计划组成。达到根据行为和子行为概念，共享计划理论描述了一个达到共同目标的层次化计划。这也是联合意图理论和共享计划理论之间的主要区别，共享计划理论阐述了一种实现共同目标的方法，而联合意图理论仅阐述了这个共同目标。然而，联合意图和联合提议等缺少理论支持，导致

在团队合作和团队行为的推理上具有局限性。

联合意图和共享计划这两种理论广泛应用于检验和阐述合作。在本书方法中，将借鉴这两种方法，尽管由于在执行过程中协同计划和 ALICA 规划之间存在相似结构而使得共享计划最明显。

2.4 约束规划

约束规划主要强调对问题的声明和描述，然后选择合适的约束求解器进行求解。Rossi 等人[138]给出最近的一篇综述。Fruhwirth[49,50]提出一种通用框架，将约束处理规则（CHR）定义为一种表示约束的统一方式。经简化后，约束处理规则具有 Head < = = > Guard | body 的形式，这意味着如果某项约束实施统一的 Head，以及没有在 Head 中调节变量的情况下，Guard 判断为真，Head 被 Body 替换。该系统已集成到约束规划框架 $ECL^{i}PS^{e}$ [2] 中。CHR 可用于约束传播与简化，然而需与搜索算法相结合来确定满足约束满足问题的解决方案。这种搜索方法通常对进一步约束的变量进行假设，一旦假设导致可检测的不满意约束，则回溯。回溯还可扩展到回跳，即多步同时回溯以对搜索空间中更有希望的区域进行搜索[124,26]。

在布尔域中的约束满足已证明是一个完全误解问题[29,143]。后来，同样证明对于普通的约束满足问题，在有限域中也是无解的[43]。随着某些可追溯的子类被确认[27]，很难满足机器人控制，尤其是许多不同问题需要制定的需求。

从机器人研究的角度出发，在连续域上的问题分类更加有意义，由于在机器人领域中很多都是真实值，如关节位置或角度状态。通常，求解实数域基于约束的数学模型问题是不可确定的[135]，然而解可以近似。寻找合理精确度的近似解之间的误差是不可能的。约束满足问题的最普通的实例是所有自由变量的一阶方程，无解。且该问题不可确定，因为通常一阶逻辑中满意度不可判定。

为解决更复杂的问题，Jónsson 和 Frank[80]提出了动态约束问题，其中单个约束问题以序列形式链接。邻近问题可通过限制或放松相互得到。可提出一种基于过程的推理方法来求解该系统。因此，所得到的系统可有效解决问题，给出相应的过程。在该方法下，对求解系统前所有变量必须已知的条件放松，考虑预先变量个数未知的问题。这样，无界规划任务可表示为约束满足问题。同理，Nareyek[105]提出了一种在图空间基于约束的局部搜索规划，其中每个图都表示一个可能的规划。

最近，分布式约束最优问题用于控制 MAS 系统的行为。该项目具体由 Petcu[121]负责。在该系统中，每个智能体都具有和控制一个约束优化问题，这

些问题与其他智能体的问题具有某些共享变量。通过交错式局部求解和信息交换，智能体获得一个全局解。通常，智能体对于该问题都没有全局解。

总之，约束规划是一种非常简洁且数学意义清楚的问题表示方式，这也是将约束规划技术集成到本研究中的主要动机。然而，从约束规划角度下的软实时来看，在考虑具有固有动态性的这些域中的一类问题尚未解决。

第3章 相关研究工作

现已有多种不同方法来描述智能体行为，从根据 Bratman[12] BDI 模型的 STRIPS 规划语言[45]和第一种推理形式，到 Dasani 等人[35]提出的2APL 等现代语言。接下来，将简要概述这一广阔领域，并讨论与本书编程语言相关的其他不同方法。

3.1 行为演算

STRIPS[45]是第一种允许智能体推理其行为的推理。尽管 STRIPS 受限于仅采用命题式前提条件和效果来描述行为，但该方法非常有效。McCarthy 和 Hayes[100]首先对此进行研究，并提出了相关基本问题，如框架问题，由此推动了情境演算的发展[133,134]。在该理论基础上开发了一系列的规划语言。最早是由 Levesque 等人[98]开发的 GOLOG，随后产生了许多改进版本，如允许并发处理、外源性行为反应和中断的 ConGolog[37]。在后来出现的著名的 IndiGolog[57]中还增加了对规划和搜索的支持。

同时，Thielscher[166]在 Holldobler 和 Schneeberger[72]的基础上提出了第二种演算方法，即流演算。该算法更侧重于描述环境的动态事实，而不是改变环境的行为。Thielscher[167]基于流演算开发了一个称为 FLUX 的编程框架。FLUX 主要考虑部分知识的表示和知识的更新。这些都是通过约束来实现的。

Kowalski 和 Sergot[88]提出的事件演算是另一种非常有前景的方法，该方法允许以时间间隔进行推理，而上述两种演算方法都仅考虑离散状态。有关详细介绍请参见参考文献［151］。遗憾的是，据我们所知，还尚未有基于事件演算的面向智能体的完整语言。

所有这些语言都侧重于智能体知识的表示、行为效果以及如何根据这些知识进行推理。相反，ALICA 则侧重于旨在解决特定问题或处理特定情况的策略的表示。而且，ALICA 是一种以团队为中心的语言，而上述语言通常都只针对单个智能体的情况。因此，从行为演算的角度来看，ALICA 可看作一种与 GOLOG 或 FLUX 相结合的程序表示。同理，从 ALICA 的角度来看，基于行为演算的语言将弥补需要行为推理的缺陷，这是因为 ALICA 特意使得环境表示更开放。

3.2 BDI 语言

根据 Bratman[12]（见 2.1 节）提出的 BDI 模型开发了一系列成功的智能体语言。而 ALICA 可看作一种没有显式表示期望或目标的 BDI 语言，这是由于其着重于将规划看作意图。接下来，将介绍与 ALICA 相关的一些具有影响力的 BDI 语言：

3APL 3APL 是一种面向智能体的编程语言[71,34]，目的是对具有认知能力的智能体建模，并对认知机器人进行高层控制。ALICA 与 3APL 具有很多相同的概念，如信念基的定义和将目标解释为“要完成的目标”，这并不是声明性描述，而是通过一个朝向实现目标的规划。然而与 ALICA 不同的是，3APL 还便于明确规范目标。在 3APL 中引入了规则集和信念以允许在目标和规划层面上进行推理。在此，ALICA 通过一种与 3APL 完全相同的过渡系统方法来定义其操作语义。实际上，在 3APL 中，用于纯语言元素的过渡系统和用于指定智能体控制结构的元语言的过渡系统是不同的。而在 ALICA 中，并没有严格区别，因此两种过渡系统可合并。尽管 3APL 的实现支持以一种与 FIPA㊀[47,46]兼容的方式进行通信，但 ALICA 所支持的多智能体显式规划不能在 3APL 中表示。ALICA 还可通过更具表现力的约束库来扩展运行时替换的概念，这是 3APL 中智能体配置的一部分。

2APL 在 3APL 之后，Dastani 等人[35]提出了 2APL，其具有如异常处理、修复机制和语言接口等特点的各种编程结构。然而，2APL 没有从全局角度对多智能体规划建模的特点。实际上，需要设计单个智能体的规划以使得其可与其他智能体通过显式消息进行交互。

AgentSpeak(L) AgentSpeak(L) 允许类似于逻辑程序来指定 BDI 智能体[129]。Rao 确定了 BDI 系统实现与理论之间的差别㊁，并试图通过引入抽象 BDI 系统实现的 AgentSpeak(L) 来克服上述问题。AgentSpeak(L) 是一种基于事件和行为的受限一阶语言的编程语言。但遗憾的是，AgentSpeak(L) 不适用于多智能体规划建模。值得注意的是，AgentSpeak(L) 可由 3APL 模拟仿真，并嵌入到其中[69]。

KARO KARO 并不是一种编程语言，而是一种基于动态逻辑的智能体逻辑。然而 Hindriks 和 Meyer[70]提出一种直接与逻辑相关的编程语言。在此，认为

㊀ 智能实体智能体基础。

㊁ Wooldridge[181]也独立确定了同样的语义问题。

动态逻辑的方式仅限于在行为并发且延续一定时间间隔的机器人领域中使用。由此，机器人场景可能更容易描述与时间间隔有关的理论，如上述的事件演算[151]。

3.3　规划执行语言

从更实际的动机来看，规划执行语言并不是按照一个理性智能体的方法，而通常是提供一个规划的执行层以及指定该执行层的语言。而这也是 ALICA 的目的所在，通常在 BDI 语言的某种意义上，并没有提供修复失败规划的多种手段。另外，也没有像 ALICA 和 STEAM 语言那样与合作紧密集成[163]（见 3.4 节）。

PLEXIL　最著名的规划执行语言之一是 NASA 的 PLEXIL[40]。PLEXIL 完全面向于确定性执行，并在其主要领域得到良好应用，即半自主空间飞行器，如卫星。将任务看作一种树形结构并提供同步执行语义。树中所有节点都并行执行，并通过共享变量进行通信。PLEXIL 并不提供任何多智能体语义，其细粒度的编程语言可分配变量并调用函数库。相比而言，ALICA 中的最小执行元素是行为，这本质上是图灵完全的程序。

SMACH　属于规划执行语言范畴的另一种著名框架是 Bohren 和 Cousins[11] 等人提出的 SMACH，这是一种在 ROS 中的任务级合作和执行的库[127]。将其作为一种高层规划系统和底层行为原语（基元）之间的中间层任务执行。尽管 SMACH 也支持其他执行策略，但主要还是支持层次状态机。

SMACH 所采用的实际内部结构，即允许在 ROS 框架中并发和服务调用的层次状态机，使得 SMACH 是在所有讨论的语言中与 ALICA 程序内部结构最相似的语言。然而，SMACH 仅针对个体机器人，并在单个系统中各元素之间提供协调隐喻。一方面，ALICA 主要针对机器人团队，提供一组层次化的状态机。另一方面，ALICA 通过效用函数和基于约束的团队行为建模来进行任务分配。

XABSL　XABSL 语言是由 Lötzsch 等人[99] 提出的一种可作为规划执行语言的行为建模方法。XABSL 语言通过状态和选项的层次化结构来描述智能体行为。Zweigle 等人[187] 将类似于 Petri 网的结构扩展到分析多智能体之间的交互。图形化建模的 XPlM 网已编译到 XABSL 树中。尽管 ALICA 中也是以层次化交互网络的总体思路为主，但与 XPlM 网存在根本不同。最主要的是，ALICA 通过其能力来描述智能体，并通过角色和任务来实现规划和智能体之间的一个两层抽象，这样就易于定义依赖于所涉及智能体异构能力的复杂合作规划。而且对任务分配和约束，ALICA 将基于状态的行为描述与效用函数相结合。前者可实现高度动态下仍保持稳定的自适应团队行为，而后者以部分一阶逻辑来扩展该语言，使之可表示实数范围内的复杂关系。因此，ALICA 比简单的基于状态的层次化方法性

能更优。

3.4 团队合作

根据2.3节中介绍的联合意图和共享规划等团队合作理论，产生了多种实现方法。在此，简要重点介绍其中最主要的几种。

GRATE* Jennings等人提出的GRATE*系统基于联合意图理论。通过共同责任的概念，GRATE*提供了一种基于规则的建模方法来实现合作，而共同责任的思想也是基于联合意图理论的。然而GRATE*主要面向智能体及其之间的通信均可靠的工业设备，这样就利用核心概念来组织构建联合行为，并在执行相应动作之前采用广泛的通信协议。

STEAM STEAM（Shell for Teamwork）[163,125]是建立在联合意图理论和共享规划理论基础上并试图克服其缺点的。基于联合意图，STEAM建立与第2章介绍的共享规划理论平行的层次结构。因此，STEAM方法通过建立和维持联合意图来实现，并利用共享规划在复杂任务中形成团队意图。

ALICA与STEAM非常相似，并借鉴了其中某些思想，同时还借鉴了联合意图理论和共享规划理论。正如STEAM，ALICA构建了涵盖整个团队及其子团队的协作行为的团队规划层次结构，并提供了团队智能体的分配机制，来确定团队成员执行跟踪动作的需要。ALICA还借鉴了联合意图理论，尤其是在同步操作定义中（见5.13节）和在需要通信故障时。然而在ALICA中，通信故障是在周期性消息中隐式表现的（见7.5节）。

与STEAM相比，ALICA智能体通常在执行合作目标之前无需建立联合意图。实际上，每个智能体对队友决策进行估计并根据这些估计进行相应动作。检测个体决策之间的冲突并利用团队成员内部状态间的周期性通信来解决冲突。尽管STEAM提供了选择性通信以及跟踪团队成员意图的方法，但本书认为对于高动态域和时序要求严格的应用，在联合行为开始之前应忽略建立或估计共同承诺的严格要求。在ALICA中，智能体不断进行决策和行为直到产生相互矛盾的信息，这样似乎更适用于这种应用。而且，ALICA提供了相关的语言元素来强制执行一个明确约定，由此对需要时间严格同步的行为产生联合意图，如合作搬运一个物体。另外，由STEAM实现的团队智能体分配以及操作者对团队的分配（对实际团队行为进行封装）对高动态域而言过于固定。例如在机器人足球比赛中，若一个作为防守队员的机器人控制着球且比赛现场情况允许的话，该机器人也应能承担进攻队员的任务。为便于实现这种行为，在此提供了一种略显不同的角色定义，并结合任务以及任务偏好的概念。与STEAM不同，ALICA不依赖于团队领导者，而STEAM对不同目的假设一个团队领导者。实际上，ALICA只有

在检测到一个长期冲突时才恢复为基于领导者的决策。由此，实现在运行期间切换合作协议。

项目“Machinetta”就是基于 STEAM 实现的[142]。为提供一种团队合作框架的轻巧便捷实现方法，Machinetta 采用代理概念来构建一个团队合作封装模型的可重用软件包。每个代理都与单个域智能体紧密配合，来表示团队中的智能体。

STEAM 只是在合作理论与实践之间搭建了桥梁，而并不是一个机器人团队的完整实现。STEAM 提供了一种推理或建立团队合作的机制，但没有详细描述规划或操作者的内部关系。STEAM 及其实现 TEAMCORE[126] 仅假设反应或本地规划，并不支持顺序和/或并行行为的实际“程序”规划，同时也并没有真正指定一个智能体的内部控制周期。在这方面，智能体编程语言对 ALICA 的设计有所启发，尤其显著的是 3APL[71] 和 2APL[35]。随后，Tambe 等人定义了一个基于 TEAMCORE 的面向团队的编程框架[164]。在此，单个智能体通过 TEAMCORE 代理在不同编程语言中实现合作。该方法本身很好地实现了一定距离内异构可编程智能体之间的协调合作，但是并没有进一步解决团队机器人所面临的动态域问题。

CAST　CAST[183,184]（Collaborative Agents for Simulating Teamwork）是一种基于共享规划理论的团队合作框架。CAST 侧重于动态环境的灵活性并通过预测团队成员需要何种信息来进行主动信息交换。其中，利用 Petri 网来表示团队结构以及团队合作过程，即将要执行的规划。这种表示方式避免了信任推理的计算复杂性，与 ALICA 中所用的有限状态机非常相似。另外，CAST 中所采用的动态角色选择也与 5.13 节中将要讨论的动态任务分配规则非常相似。

然而，相比于 ALICA 中基于智能体、角色和任务的两层抽象，CAST 中对角色表示的表现力较差。另外，当团队成员失效时，CAST 无法对其进行补偿。最后，在参与任务完成的智能体个数有界（XOR 运算符）的情况下，CAST 需要在执行动作之前先进行通信。

ALLIANCE　Parker[118] 提出的 ALLIANCE 架构旨在提高移动机器人团队合作的容错性。建立一个机器人的合作框架，并在没有任何集中控制下自适应故障。类似于 ALICA，ALLIANCE 依靠周期性广播通信来实现该目标。与 ALICA 不同的是，ALLIANCE 不具有可表示复杂协调行为的面向团队的高层规划。实际上，ALLIANCE 定义了一组行为，当其他行为休眠时，执行某个行为。根据急躁和默许变量值的概念，机器人动态选择合适的行为集。每组行为由与其他组动机行为相互关联的动机行为控制。尽管该方法可适应动态变化的情况，但从该建模角度而言，其本身并不适合于具有大量因果和时间依赖关系的复杂规划。尤其是，将规划算法产生的局部有序规划转化为似乎不可行的相应 ALLIANCE 程序。

在处理在一个具有噪声以及可能存在竞争对手的实时动态环境下的合作问题时，Reis 等人[132]确定了反应与合作之间的权衡关系。为解决上述平衡，提出通过引入一个扁平层次结构来区分称为策略的情况和反应的情况。在某种意义上，ALICA 通过允许任意深层嵌套的规划对此进行扩展，以使得在每层可利用阈值单独调节反应程度，并对任务重分配进行相似性度量（见 5.8 节）。另外，Reis 等人还区分了位置和角色的不同。位置表示一个智能体在队列中的实际位置，而角色决定了智能体的行为。该方法已成功应用于机器人足球仿真比赛中。

然而，上述所采用的角色交换算法 DPRE（Dynamic Positioning and Role Exchange，动态位置和角色交换）仅允许两个智能体之间的角色交换，并没有考虑一个全新的分配。由于 DPRE 在决定是否与其他智能体角色交换时采用潜在效用增益，两个角色切换产生冲突以及相应的效用测量相互关联时，并不清楚会发生什么情况。另外，由于该方法认为所有智能体均同构，并将队形和位置紧密集成到框架中，因此无法很好地应用到机器人足球领域。最后，该方法不提供任何规划的内部结构，除了角色交换之外也没有任何修复机制。

在此讨论的团队合作方法都没有通过约束满足和优化问题对行为进行建模。据我们所知，ALICA 是机器人领域团队合作的第一个完整框架，其中，该声明描述可作为一个完全集成的语言元素。由于 ALICA 解决运行期间动态形成的问题，鲁棒地协调团队中的解决方案，并在必要情况下利用团队的整体计算能力来解决难题，因此这是 ALICA 对团队合作方法最重要的贡献。

3.5 任务和角色分配

关于任务和角色分配，目前已有广泛研究，无论以何种方式来讨论这些方法都已超出本书范畴。近来，Campbell 和 Wu 给出了精心研究的综述文献[20,21]。Gerkey[55]对任务分配中的不同问题进行了分类，从而也对角色分配算法进行了分类。之后，对不同任务分配问题建立了一套分类系统。大体上，根据三种特性来对多机器人的任务分配进行分类：

单任务机器人（ST）和多任务机器人（MT）：表示一个机器人是否可以同时执行多个任务。在 ALICA 中，任务分配问题假设是单任务机器人，然而由于 ALICA 程序的层次特性，实际上机器人或团队机器人可并行执行不同规划，因此 ALICA 中所反映的全局问题可由多任务机器人处理。类似于 Gerkey 的假设，在多任务机器人情况下，决策相关的实体，如效用和前提条件都必须独立无关。

单机器人任务（SR）和多机器人任务（MR）：表示一个任务仅需要一个机器人还是多个机器人。ALICA 中的基数用于表示在特定背景下执行每个任务所

需的机器人个数，由此可描述一个多机器人任务问题，并由于基数可被看作时间间隔，因此某些任务，甚至某些机器人都是可选的。

瞬时分配（IA）和延时分配（TA）：表示是否需要考虑规划今后的任务，或对于未来可能的任务没有更多信息。由于环境的动态性，通常 ALICA 用于处理瞬时分配。然而原则上，ALICA 中用于估计任务分配的效用函数可与任何信息相结合，如未来可能的分配。

Gerkey 表明 ST-SR-IA 问题可将其看作最优分配问题来处理[51]。另外，还表明与本书工作最相关的 ST-MR-IA 问题以及 MT-MR-IA 问题都是 NP 问题（非确定性多项式问题）。Nair 等人表明，在考虑未来分配时，寻找一个最优分配的问题是 NEXP-complete 问题[104]。

在上述工作中，都主张通过引入角色选择和角色执行动作并采用 POMDP（Partially Observable Markov Decision Processes，部分可观测的马尔科夫决策过程）对角色分配进行建模。如果具有必要的概率估计并应用 MDP 假设，该方法可产生最优决策。ALICA 以一种更为通用的方式来使用效用函数。在没有概率估计或正确的马尔科夫模型情况下，这可表示决策理论实体（如 POMDP）或更简单的启发式估计。

Lerman 等人[94]对动态任务分配问题进行了深入研究，其中机器人需要不断重新评估其决策，并重新分配到其他任务中来适应新的情况。他们主张采用一种数学模型来分析特定的多机器人系统，这类似于大量参数集下广义仿真的思想，然而这种模型并不能用于处理由具有传感器噪声的异构机器人组成并同时处理许多不同环境信息的实际机器人团队系统。

当然，任务分配问题也可通过机器人足球比赛来解决。Vail 和 Veloso[170]提出了一种类似于 ALICA 中任务分配的方法，其中机器人根据共享信息局部计算任务分配结果，并按照该结果进行动作。然而这种任务分配算法要处理有序列表中的全部任务，并对每个任务分配一个具有最高效用函数的机器人。通过以一个合适的效用函数对所有任务进行排序，该过程可嵌入到 ALICA 的任务分配中。另外，为保持团队中的任务稳定，还指定了一个任务重新分配的最小时间间隔。该时间间隔一般为几秒的数量级。在 ALICA 中，为执行重新分配，需指定一个大于效用偏差的阈值。另外，利用一个相似性度量来区分变化很小的重新分配（如交换两个机器人的任务）以及改变整个分配的重新分配。

Weigel 等人[178]利用效用函数进行动态角色分配，即测量与其他机器人独立的每个机器人—角色对的效用，由此创建一个鲁棒且高效的角色分配算法，然而该方法局限于独立的效用测量并仅针对单任务单机器人情况。

机器人足球比赛中的其他合作和分配方法经常采用一个集中模块来进行某些决策。例如，Lau 等人[93]采用一种分布式的角色分配，但集中计算位置分配。

对于上述问题，Lau 等人实质上是采用了一个结合优先级列表的贪婪算法。

Wang 等人[175]提出了另一种算法，其中机器人根据少数者博弈模型切换到最少表示的角色[22]。由此，可根据之前选择的结果来动态适应角色选择策略。该研究成果具有广阔前景，但产生的性能有赖于所考虑历史的长度，并且算法空间复杂度与历史步长成指数关系。

3.6 评估协议和冲突消解

如果没有进行通信，STEAM 可推理通信成本与协调质量风险之比[163]，然后提供维护和保持团队合作所需的修复运算符。尽管这种方法很有吸引力，但在大多数实际应用中，该方法所需的确定通信成本和失败风险的必要参数不能事先准确确定。实际上，只能进行大致估计。

还可利用规划识别来检测，然后修复团队合作中产生的冲突，正如 Kaminka 和 Tambe[82]通过采用社会周至监测（Socially attentive monitoring）来实现。可利用规划识别来扩充本书所提方法，而且可能会提高整体性能，但这已超出本书范畴。

Kaminka 和 Frenkel[81]提出的 BDI 架构的 BITE 可允许同步动作以及对任务分配建模，这与本书所提方法相似。BITE 为所有问题都提供可互换的决策协议，以使得设计者可对每个特定问题选择最佳协议。以类似的方式，ALICA 可允许通过条件的弱同步，或通过同步元素实现更多通信的增强同步（见 5.13 节）。另外，当检测到持续冲突时，ALICA 可自动切换任务分配的决策协议（见第 6 章）。而 BITE 不支持决策协议间的自动切换，而且在所有情况下，BITE 都要在行动之前建立通信协议，而 ALICA 会在通信之前就动作，因此 ALICA 更注重动态环境下的反应。在 ALICA 中的重要假设是情况的变化会比信息交换更快，因此及时动作至关重要。

最后，与上述讨论的所有方法不同的是，ALICA 还支持由实值变量下的约束系统进行团队行为建模并提供一种机制来协调感知较弱情况下的结果。即由于所假设的环境动态性，协作并不是要准确达到 $\boldsymbol{R}^n$ 中的同一矢量，而是通过协作来建立一种接近相似值的趋势。协作方法将在 10.3 节中详细讨论。

3.7 任务模型

根据将复杂任务分解为较小任务来简化原始问题的思想引出了层次化任务网络（Hierarchical Task Networks，HTN）的研究[140]。HTN 已广泛用于规划算法，根据动作分解运算符，将高层任务逐步分解为细粒度结构。这种规划步骤的结果

可表示为一种 ALICA 规划结构。

后来，HTN 原始结构又启发了 TAEMS 模型（任务分析、环境建模和仿真）的产生[74]。TAEMS 是一种非常丰富的多智能体任务模型，其中每个任务都有一个完成期限，并且还可表示任务间各种不同的相互关系。最重要的是，TAEMS 采用分布式目标树来表示智能体的行为，与 ALICA 不同的是，没有假定常用的程序体系结构。实际上，智能体在运行期间或许可发现与其他树的相互关系。在这方面，与 TAEMS 相比，ALICA 更类似于 STEAM，因为都假定了一个常用的程序体系结构。而且与 TAEMS 中任务间的各种相互关系不同，ALICA 利用有限状态机并支持状态机之间的相互关系，如允许转移到其他状态机中其他智能体所处的状态。由此，ALICA 比 TAEMS 具有更多的操作姿态。随后，TEAMS 利用通用部分全局规划（GPGP）进行扩展[95]，产生一个面向规划的方法，主要侧重于分布式环境中并发任务的调度。

3.8 基于约束的建模

Chalmers 和 Gray[23] 表明如何将 BDI 慎思理解为约束求解问题。FLUX[167] 以部分接地项（partially grounded term）的形式来表示部分知识，其中约束表示其余额外知识。

Ooi 和 Ghose 实现了智能体描述中的第一个约束集成[117]。后来 Dasgupta 和 Ghose[33] 对此进行扩展，该方法可集成智能体要优化的目标。这些方法都基于 BDI 语言 AgentSpeak(L)[129]。

上述工作表明在高层智能体描述语言中的集成约束可极大提高效率和表现力，然而在 Ooi 和 Ghose[117] 的工作中，约束只能用于选择或产生基于意图的规划。而本书对动作或行为参数进行约束，由此可利用约束以一种更具体和直接的方式来描述行为。

相对于这些对 AgentSpeak(L) 的扩展，本书方法从全局角度对多机器人团队进行约束建模，允许约束问题的层次化分解，由此更易于对单个智能体的子问题进行求解，在此假设约束动态改变，并在求解过程中集成了合作（见第 8 章）。

De Schutter 等人[38] 提出了一种对个体机器人特定复杂任务进行建模的通用方法，这需要在运行期间采用控制器结构。这些工作都主要侧重于控制，而本书提出了一种无需控制模型的通用的非线性约束。在 ALICA 中，具体的控制器由执行行为定义。

在参考文献 [38] 中，约束仍局限于控制器输出和关节坐标的等效约束。后来，Decré 等人[39] 对所得框架进行扩展，使得约束问题更加明确。该框架适用

于具体问题的目标函数和包括不等式的约束。另外，还增加了非瞬时约束，即随时间变化的约束。然而问题类仍存在凸集，且不能支持非线性约束的任意布尔组合。iTasc 和 ALICA 中基于约束的控制的根本差别在于 iTasc 的控制器只能在违反约束时作出反应，而在 ALICA 中，采用控制器之前就已对约束进行了求解。因此，ALICA 的求解时间可能会比 iTasc 更长，但是可从全局角度来考虑控制任务。

第 2 部分　命题式 ALICA

第 4 章　语　　法

本章将逐步介绍一种命题式 ALICA（记为 pALICA）的语法元素。第 5 章将介绍相应的语义。在第 8 章，将该基本语言扩展到一阶情况。直观上看，该基本问题可非常简洁地表示为一组智能体需要以一种协调一致的方式朝着一个共同目标采取行动。为实现该目标，团队应遵循一组确定行为过程的指令，此时需要不断监视团队的行为进展来检测和避免可能存在的缺陷或对已经发生的问题进行补偿。

由目标描述推导出一组指令集的过程称为规划，然而规划并不是本书的重点。实际上，是要设计一种可制定指令集的语言。另外，还要考虑这些指令的有效性以及协调执行。

例如由两个机器人组装一台小型结构的情况。规划算法将计算一组机器人以抓取部件、运动并将部件安装到结构上的顺序来执行的指令集。该语言可允许规划者表示必要的概念，如行为动作之间的时间和因果依赖关系。例如，如果机器人 a 负责组装部件 p，则机器人 b 可能需要等待该动作完成，这样才能组装与 p 相关的其他部件。每个机器人的执行层对这些指令解释并执行。同时，不断监视行为进展情况，并在上一个动作成功完成后调用每个个体动作。另外，还应对失败作出反应和补偿，例如机器人没有成功抓取一个部件，或甚至某个机器人完全失效。最后，还需处理必要的通信和协调，这样才能使每个机器人了解各自的进度。

可自主行为的智能体在处理时具有一组特定的基本动作。编程语言应能在任何环境下的每个时间点确认各个智能体所采取的正确动作。为达到该目的，需要一种关于环境和条件的信念（belief）表示方法。假设已具有足够的逻辑 $\mathcal{L}$ 在语言 $\mathcal{L}$(Pred，Func）中表示上述条件。尽管假设了一组谓词 Pred 和一组函数符号 Func，但在该语言中不需要，这样 $\mathcal{L}$ 具有某种一阶的意味。然而，$\mathcal{L}$ 完全可以编译到命题式逻辑中。

另外，为描述场景中的特定智能体，将包含所有可能在团队中参与的智能体记为集合 $\mathcal{A}$。$\mathcal{L}$ 和 $\mathcal{A}$ 这两个实体构成一个 pALICA 程序的域签名（domain signature）。

定义4.1 一个pALICA程序的域签名（$\mathcal{A}$，$\mathcal{L}$）包括：

● 构成合作团队的智能体集合$\mathcal{A}$；

● 描述具有一组谓词Pred和一组函数符号Func的智能体信念基的语言$\mathcal{L}$(Pred，Func）中的逻辑$\mathcal{L}$。

用$\vdash_{\mathcal{L}}$表示$\mathcal{L}$中定理证明演算及其相应算法，即$\mathcal{F}\vdash_{\mathcal{L}}\phi$表示从一组公式$\mathcal{F}$中推导逻辑结果$\phi$的算法$\vdash_{\mathcal{L}}$。在第8章中，将该基本语言进行扩展以集成约束满足问题，并更详细地讨论了相应算法。在此，假设已具备$\mathcal{L}$的证明算法。如果逻辑清楚，就省略之而仅记为$\vdash$。在命题式ALICA（pALICA）中，没有使用可能具有自由变量的公式。用$\mathcal{L}_S$表示$\mathcal{L}$(Pred，Func）的结果集。因此，类似于SMT理论㊀，pALICA程序可解释为一个命题式公式，其中变量又形成另一种理论，即$\mathcal{L}$。

4.1节，在一个智能体中引入基本动作元素——行为。相对于传统的行为演算，ALICA中的行为具有显著的持续时间，这通常仅是松散有界的。4.2节中继续介绍高层概念——规划，即根据最小行为来构建一个结构。其余部分讨论了其他概念，如同步操作、角色和语法规范的程序。

4.1 行为

在真实场景中，实际机器人完成动作（如到达某一特定房间）都需要大量时间，且是非确定性的。也就是说，机器人可能失败，或即使成功可能也会有不同结果。例如，由于传感器的噪声和执行器的不精确，多次重复到达厨房的动作几乎不可能使得机器人到达完全相同的地点。

这就产生了编程语言中的第一个元素，即称为行为的封装在有限集$\mathcal{B}$中的底层不可分动作。通常，行为只能在特定条件下执行，如去打开一个已经开着的门没有任何意义，或甚至很危险地到达一个不可测的道路交叉口。因此，在执行每个行为时都需要一个前提条件：

$$\text{Pre}: \mathcal{B} \longmapsto \mathcal{L}_S$$

在此，用Pre(b）表示行为b的前提条件。另外，由于执行行为需要一定的时间，因此这可能也是在整个执行期间应保持的条件，称为运行条件㊁：

$$\text{Run}: \mathcal{B} \longmapsto \mathcal{L}_S$$

最后，在大多数情况下，行为意味着对环境产生某种程度的改变。换句话说，一个智能体执行这些行为是为实现一个后置条件：

㊀ 关于SMT求解，请参见Nieuwenhuis相关研究[112]。

㊁ 运行条件涉及其他演算中的不变量。然而与传统非变量不同，运行条件必须初始化为真。另外，运行条件可改变其值，如由于环境中不可预见的变化，会引起相应的故障。

$$\text{Post}: \mathcal{B} \longmapsto \mathcal{L}_S$$

根据这三种类型的条件，就可能形成并解决规划问题，这类似于 STRIPS 中的命题式问题[45]。与行为演算不同，在此并没有对框架问题提出一个解决方案[100]，同时也没有定义更新方程。实际上，本书工作是依赖已有的解决方案（如参考文献［92，134，150，168］），并根据具有任何信念或知识表示以及相应更新语义的 ALICA 可互换操作。

根据这种思路，对行为完成标记为成功或不成功。也就是说，行为本质上是一个或多个算法实现的软件组件，如控制器或各种搜索范式，并不需要依靠信念基的实现来涵盖必要的概念，也不需要具有推理算法的能力来推断后置条件。从语义上来看，行为需要两个谓词，即 Success(b) 和 Fail(b) 在 $\mathcal{L}$(Pred，Func) 中表示，这两个谓词可由行为 b 来设置。值得注意的是由于$\mathcal{B}$有限，这两个谓词应编译到命题式逻辑中。

然而规划算法通常依靠后置条件来寻找一条到达目标状态的合适路径。在某些场景中，可采用仿真器来试运行，但这通常并不可行。因此，规划需要成功信号和后置条件的等效逻辑。

定义 4.2　规划公理

$$\Sigma_{\text{plan}} \overset{\text{def}}{=} \text{Success}(b) \leftrightarrow \text{Post}(b)$$

然而这并不需要推理算法 $\vdash_{\mathcal{L}}$在所有行为成功的情况下来证明后置条件。尽管对于任意信念集 B ，必须满足

$$B \cup \{\Sigma_{\text{plan}} \wedge \text{Success}(b)\} \vdash_{\mathcal{L}} \text{Post}(b)$$

这实际上可归结为一个假言推理（前件肯定式）的应用。

4.2 规划

给定不可分割的行为及其注释，需要一个结构来形成更加复杂的方案或规划。该结构构成编程语言的核心。一般而言，这是描述为实现某一目标、保持某一条件或控制一个动态系统所要执行的动作。描述一个规划的最简单方法是一个行为顺序执行的列表。这种方案或策略有多种不同表示方法，是对上述简单描述的扩展。

动作选择的一个主要表示方法是决策树，根据环境的某种特性和与每个叶子节点关联的行为，在决策树中的每个节点进行决策。决策树及其更简单的形式——决策列表已在机器学习领域进行了深入研究。请参见 Quinlan[128] 和 Murthy[103] 有关决策树学习的研究工作以及参考文献［156］中表示单个智能体行为的决策列表方法。

单纯的决策树只能针对反应式确定性行为。利用概率原则可将上述扩展到非确定性情况，由此产生更适合于分类经典增强学习技术的表示方法。对于更复杂的任务，智能体需记住其部分历史。以一种受限方式来保存历史的简单方法是采用有限状态自动机，这样可比决策树的表示更加紧凑。另外，还需注意的是任何一个决策树都可简单地表示为一个有限状态自动机。近年来，采用任务网络和层次化任务网络（HTN）来表示智能体行为，主要是针对规划问题[140,106]。本书的目的是寻求一种更具表现力的编程语言，其具有表示替代、循环和例如同步以及并发任务之间依赖关系的团队合作等特点。

因此规划的基本概念与有限自动机紧密相关，不仅可形成动作序列，还可循环和条件执行。规划集$\mathcal{P}$中的每个规划 p 都是一个有关状态及状态转移的结构。在 pALICA 程序中所有状态的集合记为$\mathcal{Z}$。状态转移是直观地表示一个智能体何时以及如何从一个状态切换到另一个状态。因此，每个状态转移 t 表示两个状态之间的关系，并是$\mathcal{L}$中的一条语句。pALICA 程序中的状态转移定义为$\mathcal{W}\subseteq\mathcal{Z}\times\mathcal{Z}\times\mathcal{L}_S$。对于每个规划，状态转移和状态之间的关系构成了一个有向图，其中转移为边，状态为节点。

状态所包含的直观含义是类似于 petri 网中的库所，处于某个状态的智能体应执行该状态所必须完成的事情，即执行一个或多个行为。函数 Behaviours：$\mathcal{Z}\mapsto 2^{\mathcal{B}}$定义了一个状态中所包含的行为，并由处于该状态的智能体执行。

一个有限自动机具有一个表示从此开始计算的初始状态，然而仅仅一个初始状态不足以反映多智能体系统的行为。实际上，可设置多个初始状态，对于每个智能体子集，每个初始状态都各自定义一个起始点。由此，自动机也相应地分为子自动机，各子自动机之间无需关联，这样就可描述相互依赖的多个智能体的行为。5.13 节中介绍了一种描述该依赖关系的方法，由此实现一个规划中不同行为的协调合作。现在，引入一种任务概念，即指向规划中的初始状态，并确定其中的子自动机。程序中所有任务的集合记为$\mathcal{T}$。值得注意的是，并不是刻意使各个任务所确定的子自动机不相交。子自动机不相交是有利的，因为在这种情况下，每个状态明确地确定了一个任务。这可用来限制通信成本（见 7.5 节）。

部分函数 Init：$\mathcal{P}\times\mathcal{T}\mapsto\mathcal{Z}$将任务和规划映射到一个初始状态，如 Init$(p,\tau)$ 表示规则 p 中任务 τ 的初始状态。这使得在考虑规划集时，任务与随后所用的具体规划，特征相互独立。考虑到任务其实是确定规划中不同部分的方法，就可分配智能体完成规划中的不同部分，然而某些任务仍需要多个智能体。例如，对于单个智能体而言，某个物体或许太重而无法移动，或区域太大而无法在允许时间内搜索完。因此，就可采用多个智能体来执行规划中的相同任务。执行一个任务的智能体个数可能受某些约束条件的限制。利用部分函数 ξ：$\mathcal{P}\times\mathcal{T}\mapsto\mathbb{N}_0\times(\mathbb{N}_0\cup\{\infty\})$ 可限定智能体的个数，如 $\xi(p,\tau)=(n_1,n_2)$ 表明在规划 p 中执行任务 τ 需要至少 n_1

个，最多 n_2 个智能体。

某些规划意味着要保持某种条件并无限运行。然而另外一些规划是目标导向的，一旦达到期望的环境状态则终止运行。为此，在此将终止状态也包含在规划结构中。设 States(p) 表示规划 p 的状态，则状态 Success(p) ⊆States(p) 表示成功完成的终止状态。一旦智能体达到该状态，则表明已成功完成该任务。同理，用 Fail(p) ⊆States(p) 表示失败的终止状态集。当智能体达到一个失败状态时，表明没有完成任务。在 5.5 节中，将详细讨论成功的结束。5.13.2 节中将介绍处理失败的可能性。

根据所定义的元素，就可表示某些复杂行为，如图 4.1 中服务机器人领域的示例。等待规划包含两个任务：一个是服务于餐桌的一组机器人；另一个是负责从门口到服务区域的单个机器人。

图 4.1　规划示例：等待

然而这里有两个主要缺点：首先，由于问题比较复杂，因此针对具体解决方案，还没有采用一种抽象复杂性的方法；其次，在单个状态机中表示代替方案比较繁琐。若可能有另一种转换，需要快速适应时，则可快速地完全包括所有连通的状态机。这些问题都需要一种方法来相互比较各种规划。

因此，在此引入类似于针对行为定义的规划条件。增加函数 Pre 和 Run 来将规划集与 $\mathcal{L}$ 语句相对应：

$$\text{Pre}: \mathcal{P} \cup \mathcal{B} \mapsto \mathcal{L}_S$$

$$\text{Run}: \mathcal{P} \cup \mathcal{B} \mapsto \mathcal{L}_S$$

为便于规划，还应设定一个后置条件。由于定义了终止状态的概念，因此可将后置条件与成功的终止状态相关联。这要求对各个规划中成功实现的不同结果进行建模。同理，失败状态也要设置后置条件，这可通过一个推理模块来进行失败分析。在此增加 Post 部分函数

$$\text{Post}: \mathcal{Z} \cup \mathcal{B} \mapsto \mathcal{L}_S$$

将终止状态与后置条件相对应。在语义上，状态的后置条件与行为的后置条件有很大不同。如果达到一个终止状态，这表明满足后置条件。然而，根据智能体的内部信念，后置条件评估为真并不意味着一个智能体或即使是相应的智能体位于相应的终止状态。在 5.13 节介绍运行时语义时，将讨论上述区别。

根据所介绍的条件集合，对于一种具体情况，可对当前规划或规划过程中的假设条件进行评估。设$\mathcal{L}$(Pred, Func) 单调 ㊀，则难以表示在应满足多个前置条件和运行时条件的所有规划中所首选的那个规划。

因此，引入第 4 个元素来评估一个规划，即效用函数（utility function）。一个效用函数与一种情况相对应，由信念基，即一组$\mathcal{L}$(Pred, Func) 中的公式进行描述来定义总顺序集合。根据一种常用选择，设效用函数映射到实数集：

$$\mathcal{U}: \mathcal{P} \mapsto 2^{\mathcal{L}_S \mapsto \mathbb{R}}$$

每个规划 $p \in \mathcal{P}$都有一个效用函数，记为$\mathcal{U}(p)$，反过来又对应一组实数公式。然后，根据情况来评价规划。由于规划可能有多个初始状态，因此也可对执行规划的不同方式进行评价，即不同分配，这将在 5.8 节中讨论。

给定效用函数以及前置和运行条件，相对于某一种情况，可对规划进行比较。为允许智能体通过比较从一组候选规划中选择其中一个，在此将规划进行分类，称之为规划类型。这样的分组规划（不是必要条件）可直接实现同一目标。pALICA 程序中所有规划类型的集合记为$\mathcal{P}_\vee \subseteq 2^{\mathcal{P}}$。属于规划类型 P 的一个规划 p 也称为 P 的实现。

为促进层次结构的思想，将这些集合作为状态属性引入，这样每个状态可以与包含行为的方式一样包含一组候选规划集。函数

$$\text{PlanTypes}: \mathcal{Z} \mapsto 2^{\mathcal{P}_\vee}$$

将每个状态映射到候选规划集中的空集，这意味着每个规划类型都是并行执行的。

状态和规划类型之间的关系以及规划类型与规划之间的关系构成了一个规划有向图。4.5 节中，将该图限制为一个树。树中最顶层元素由规划 p_0确定。其中包含单个任务 τ_0和单个状态 z_0。这样，p_0就可看作多智能体程序中的主程序。在

㊀ 参见参考文献［14］确定如何将选择与非单调逻辑相关联。

此，将该结构称为规划树。

图 4.2 表明如何通过层次化抽象来简化图 4.1 中的 Waiting 规划。引入两个新的规划类型：一个是用于服务一个餐桌；另一个是用于从房间传送食物。这两个规划类型可对每个任务的复杂性进行抽象，同时也增加了重用的可能性。例如，从房间传送食物的规划也可用于其他设置。规划 GetOrder 和 BringFood 是 HandleTable 的两个实现，分别用于处理由其各自的前置条件区分的某一具体情况。值得注意的是，只有规划 BringFood 在终止状态结束，这表明成功实现对某一特定餐桌的服务。

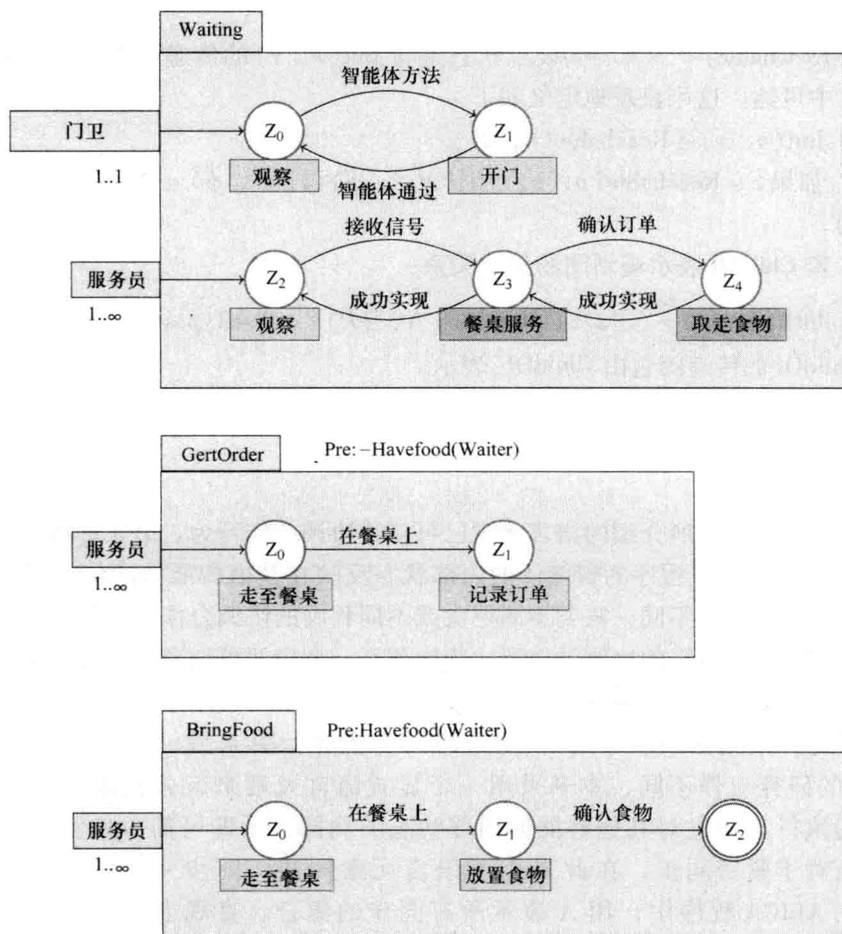


图 4.2　规划示例：层次化等待

上面所介绍的规划结构在第 5 章中将具有重要作用，因此需引入简化规划表示的一些宏：

● 利用 $\text{PlanTypes}^+(z)$ 来表示规划类型的传递性，可直观地定义为

✓ $\text{PlanTypes}(z) \subseteq \text{PlanTypes}^+(z)$。

✓ 如果 $P \in \text{PlanTypes}^+(z)$，则 $(\forall z', p) p \in P \wedge z' \in \text{States}(p) \rightarrow \text{PlanTypes}(z') \subseteq \text{PlanTypes}^+(z)$。

● Plans：$\mathcal{Z} \mapsto 2^{\mathcal{P}}$，Plans($z$)表示可在状态 z 中执行的规划集：

$$\text{Plans}(z) \stackrel{\text{def}}{=} \bigcup \text{PlanTypes}(z)$$

● Plans^+：$\mathcal{Z} \mapsto 2^{\mathcal{P}}$为规划集的传递闭包，定义为

$$\text{Plans}^+(z) \stackrel{\text{def}}{=} \bigcup \text{PlanTypes}^+(z)$$

● Reachable：$\mathcal{P} \times \mathcal{T} \mapsto \mathcal{Z}$表示状态集与 Init($p$, τ)的传递连接，即任务 τ 在规划 p 中可达。这可直观地定义如下：

✓ $\text{Init}(p, \tau) \in \text{Reachable}(p, \tau)$。

✓ 如果 $z \in \text{Reachable}(p, \tau)$，则 $(\forall z', \phi)(z, z', \phi) \in \mathcal{W} \rightarrow z' \in \text{Reachable}(p, \tau)$。

● 宏 ChildOf 表示规划间的父子关系：

$$\text{ChildOf}(p, p') \stackrel{\text{def}}{=} (\exists z) z \in \text{States}(p) \wedge (\exists P) P \in \text{PlanTypes}(z) \wedge p' \in P$$

ChildOf 的传递闭包由 ChildOf^+ 表示。

4.3 同步

到目前为止，所介绍的语言元素已可描述协调合作行为。第 5 章将讨论如何通过将执行 ALICA 程序的智能体的内部状态反映在其他智能体的信念基中来实现。然而根据场景不同，需要不同层次或不同程度的协调合作。例如，合作抬起一个易碎物体时所需的协调动作要比依次离开一个房间更加密切。在约束限制方面，应在通信延迟允许条件下尽可能地执行严格同步的动作。

因此，根据 Kinny 等人[85]提出的同步概念，引入另一个语言元素。与 Kinny 的研究工作不同，本书采用一个显式语言元素来区分松散同步与紧密同步的执行。通过对其他智能体内部状态的局部表示即可简单地实现松散同步。而对于紧密同步，在此采用了语言元素同步。同步 s 为$\subseteq \mathcal{W}$的转移集合。在 ALICA 程序中，用 Λ 表示所有同步的集合。直观上看，同步需要涉及的所有智能体都应同步地沿各自转移进行动作，即在同一时间。另外，整个智能体集合都沿上述转移动作，或根本没有动作。这需要类似联合意图的一定程度的承诺[97]。在其他编程语言中也引入了类似结构，如规划语言 MAPL中的同步语言行为[13]。

4.4 角色

规划描述了行为，即描述了如何达到某个目标。任务表示规划中的具体部分。为将智能体分配给任务，需要称为角色的第3个概念。赋予智能体相应的角色，以允许评估一个智能体对某一特定任务的充分性，并建立一个一般的团队构成。根据 Wooldridge 等人[182]的思想，ALICA 中的角色反映了一个智能体在团队中的期望功能。该功能由问题域、智能体能力和团队构成来确定。例如，配置有先进传感器的智能体可比具有中等感知能力的智能体执行更有效的搜索任务，由此产生了由 Campbell 和 Wu[20]所给出的角色和任务定义。

设$\mathcal{R}$表示一个 pALICA 程序中所有角色的集合，然后函数 Pref：$\mathcal{R}\times\mathcal{T}\mapsto[-1,1]$将角色和任务映射到区间$[-1,1]$中的一个实数，以表示对于特定角色的任务优先级。

有多种方法用于解决为智能体分配角色的问题。最简单的情况是，每个智能体都赋予一个固定角色。Gerkey[55]将由此产生的问题描述为单任务单机器人的瞬时分配（ST-ST-IA）。并可在多项式时间内求解。

更复杂的技术可适用于团队成员损坏或部分功能丧失的情况，使得每个智能体可具有多个角色或多个智能体具有同一角色，并考虑在未来事件中可能发生的问题。在这种情况下，如果考虑未来可能出现的重新分配，问题的复杂性就会变为无解[55]或甚至 NEXP-time [104]。近来，Campbell 和 Wu[21]对可能的角色分配算法进行了研究。

执行一个 ALICA 程序，需要每个智能体赋予一个非空角色集，且该集合对团队所有成员已知。本书没有对 ALICA 限制一个特定的角色分配算法，只是给出 ALICA 中关键概念的一般性描述。5.6 节将介绍一个适当的角色分配算法实例。

为根据智能体所具有的不同技能和能力来实现角色分配，在此还需另一个概念：能力。能力是对特殊技能及其某种评价程度的描述，如“机器人 r 非常快”或“该角色需要抓取能力”等非正式语句描述都是指能力。再次强调，在此并不对如何表示能力进行任何限制，但在 5.6 节中将通过介绍一种具体的角色分配算法来对此进行深入讨论。在此，用$\mathcal{C}$表示所有能力的集合。

函数 Cap：$\mathcal{R}\cup\mathcal{A}\mapsto 2^{\mathcal{C}}$将智能体与其提供的能力相对应，并将角色与其所需的能力相对应。给定智能体、角色及其各自所需和所应具有的能力，角色分配算法可将智能体和角色相匹配。然而，只有角色与智能体数量相同且每个智能体必须只能选择一种角色（ST-SR）时，才能实现明确分配。另外，不能考虑其他附加信息。因此，本书引入第3个概念：队形。给定一组特定的智能体以及可能需要决策的场景，通过提供所需角色的信息及其各自的优先级，队形可解决上述问

题。由于队形很大程度上依赖于角色分配的语法，在此留作一个开放问题，并在5.6节通过一个实例进行讨论。

4.5 良好架构

在上述章节中，介绍了pALICA的语言元素，并阐述了之间的某些关系，如规划树。现在从语法形式上来约束pALICA程序的语法，以保证这些关系的预期结构，由此产生架构良好的pALICA程序的概念。设Σ_{syn}为一个包含以下公理的集合：

- 高层规划仅包含一个状态z_0和一个任务τ_0：

$$\mathrm{States}(p_0)=\{z_0\}\wedge\mathrm{Tasks}((p_0)=\{\tau_0\}\tag{4.1}$$

- 状态最多属于一个规划：

$$(\forall p,\ p'\in\mathcal{P})\mathrm{States}(p)\cap\mathrm{States}(p')=\emptyset\vee p=p'\tag{4.2}$$

- 在不同规划中不存在状态转移：

$$(\forall(z_1,\ z_2,\ \phi)\in\mathcal{W})(\exists p\in\mathcal{P})z_1\in\mathrm{States}(p)\wedge z_2\in\mathrm{States}(p)\tag{4.3}$$

- 同步仅发生在同一个规划中：

$$\begin{aligned}(\forall s\in A)(\forall w,\ w'\in s)(\exists z_1,\ z_2,\ z_3,\ z_4,\ \phi_1,\ \phi_2)w=(z_1,\ z_2,\ \phi_1)\\\wedge w'=(z_3,\ z_4,\ \phi_2)\wedge(\exists p\in\mathcal{P})z_1\in\mathrm{States}(p)\wedge z_3\in\mathrm{States}(p)\end{aligned}\tag{4.4}$$

- 失败集和成功集是相应状态集的不相关子集：

$$\begin{aligned}(\forall p\in\mathcal{P})\mathrm{Success}(p)\cup\mathrm{Fail}(p)\subseteq\mathrm{States}(p)\\\wedge\mathrm{Success}(p)\cap\mathrm{Fail}(p)=\emptyset\end{aligned}\tag{4.5}$$

- 与某一规划关联的任务确定了该规划中的一个初始状态：

$$(\forall\tau\in\mathrm{Tasks}(p))(\exists z\in\mathrm{States}(p))\mathrm{Init}(p,\ \tau)=z\tag{4.6}$$

- 所有规划—任务对都具有一个相关的有效基数间隔：

$$(\forall p\in\mathcal{P},\ \tau\in\mathcal{T})\tau\in\mathrm{Tasks}((p)\rightarrow(\exists n_1,\ n_2)\xi(p,\ \tau)=(n_1,\ n_2)\wedge n_1\leqslant n_2\tag{4.7}$$

- 每个成功状态和失败状态都关联一个后置条件：

$$(\forall p\in\mathcal{P})(\forall z\in\mathrm{Success}(p)\cup\mathrm{Fail}(p))(\exists\phi)\mathrm{Post}(z)=\phi\tag{4.8}$$

- 终止状态没有附加子规划或行为：

$$\begin{aligned}(\forall z)((\exists p)z\in\mathrm{Success}(p)\vee z\in\mathrm{Fail}(p))\rightarrow\mathrm{PlanTypes}(z)=\emptyset\\\wedge\mathrm{Behaviours}(z)=\emptyset\end{aligned}\tag{4.9}$$

- 高层规划与所有规划相连：

$$(\forall p)p=p_0\vee\mathrm{ChildOf}^+(p_0,\ p)\tag{4.10}$$

- 每个规划最多有一个父节点：

$$(\forall p, p_1, p_2)\text{ChildOf}(p_1, p) \wedge \text{ChildOf}(p_2, p) \rightarrow p_1 = p_2 \quad (4.11)$$

- 父子关系的传递闭包是非对称的：

$$(\forall p, p')\text{ChildOf}^+(p, p') \rightarrow \neg\text{ChildOf}^+(p', p) \quad (4.12)$$

一个 ALICA 程序当且仅当满足Σ_{syn}时才具有良好架构。

式（4.1）保证在规划树中存在一个唯一的根节点。由于相对于最高层任何智能体所具有的可能状态数有限：无论是参与的还是没有参与的智能体，这将简化智能体的执行和协调合作。式（4.2）~式（4.6）将规划的内部结构限制为局部，即属于不同规划的元素不能相连。式（4.6）~式（4.8）保证了任务执行所需的某些必要元素，如任务—规划元组中的基数。值得注意的是，如基数（0，∞）和后置条件$\top$等具体元素都是允许的。

式（4.9）不允许终止状态有子节点。尽管这并不是绝对必要的，但直观上，终止状态会终止执行，因此不应再包含内部程序。

式（4.10）~式（4.12）将规划树结构限制为一个有限状态树。更具体的，式（4.10）强调高层规划p_0传递连接到所有规划。式（4.11）强调每个规划最多有一个父节点，这与式（4.10）一起意味着除了p_0之外，每个规划都只有一个父节点。最后，式（4.12）强调了父子关系传递闭包的不对称性。由于$\mathcal{P}$有限，规划树也有限。值得注意的是，根据式（4.10）和式（4.12），p_0不能有父节点。

从软件架构来看，强制执行一个树结构似乎过于严格，在具体实现上也可利用一个有向非循环图，使得规划可在不同环境下重用。运行时引擎识别不同的规划实例，并进行相应处理，就好比是不同规划。在此，将结构限制为树结构是为了避免过于复杂的符号，这在区分规划和规划实例时可能需要。值得注意的是，通过复制具有多个父节点的节点，一个有向非循环图可容易地转化为树结构，因此相对于非循环图，表现形式并不受限。

然而，在理论和实现上，规划树不能含有循环。若允许循环则需递归定义，从而增强表现力，在多智能体中递归语法并不简单。例如，在给定时间内，每个智能体都可在不同递归深度下工作，因此本书不考虑递归。

4.6 pALICA 语法元素概述

总而言之，一个 pALICA 程序仅包含少量不同的元素。首先，域签名定义了可能的智能体集合以及有关环境的逻辑表示信念。pALICA 的核心元素是规划，其是解决给定问题的形式化方法。规划包括状态及其转移，通过同步转移可增强智能体之间的紧密耦合。状态包括表示候选规划集的规划类型，另外状态还包括行为，即不可分的动作程序。由此产生一个树形图，其中节点是规划。任务规定

了哪些智能体可被分配规划中的初始状态，这样的分配考虑了每个智能体的角色。根据智能体所需和所具有的能力，为智能体分配角色。

表4.1总结了pALICA中的正式元素。除$\mathcal{L}$(Pred，Func)之外，所有集合都是有限的。如果$\mathcal{L}$(Pred，Func)编译为命题式逻辑，则也可允许程序编译为命题式逻辑。通过表4.2中所列的（部分）函数可构成单个元素。

表4.1 pALICA程序中的元素

$(\mathcal{A},\mathcal{L})$	域签名	域签名包括交互智能体集合以及表示环境的逻辑
$\mathcal{R}$	角色集	该集合包括所有智能体可分配的角色
$\mathcal{B}$	行为集	行为是不可分的动作程序，这构成与环境交互的方式
$\mathcal{P}$	规划集	每个规划描述了一个特定协调合作的行为
$\mathcal{P}_{\vee}$	规划类型集	一个规划类型是一组候选规划的集合
$\mathcal{T}$	任务集	每个任务直观描述了规划中的功能或责任，意味着由一个或多个智能体来完成
$\mathcal{Z}$	状态集	规划中的状态是行为过程中的一个动作，其中包含规划类型和行为
$\mathcal{W}$	转移集	每个转移(z_1, z_2, ϕ)表示前一状态z_1和下一状态z_2以及条件$\phi \in \mathcal{L}$（Pred，Func）的关系
Λ	同步集	用于表示同步行为所需的每个相关转移
$\mathcal{C}$	所有能力集合	能力用于匹配智能体和角色
p_0, τ_0, z_0	顶层元素	顶层规划，任务和状态分别构成程序图的根节点

表4.2 pALICA程序的结构定义

States：$\mathcal{P} \mapsto 2^{\mathcal{Z}}$	状态函数将规划与约束状态集相对应
Tasks：$\mathcal{P} \mapsto 2^{\mathcal{T}}$	任务函数将规划与相关任务集相对应
ξ：$\mathcal{P} \times \mathcal{T} \mapsto \mathbb{N}_0 \times (\mathbb{N}_0 \cup \{\infty\})$	ξ定义了规划p中一个任务τ所分配的智能体上限和下限
Init：$\mathcal{P} \times \mathcal{T} \mapsto \mathcal{Z}$	初始化函数将规划和任务与相应初始状态相对应
Pre：$\mathcal{P} \cup \mathcal{B} \mapsto \mathcal{L}_S$	Pre(p)表示规划或行为p的前置条件
Run：$\mathcal{P} \cup \mathcal{B} \mapsto \mathcal{L}_S$	Run(p)表示规划或行为p的运行条件
PlanTypes：$\mathcal{Z} \mapsto 2^{\mathcal{P}_{\vee}}$	PlanTypes(z)表示在状态z中执行的规划类型集
Behaviours：$\mathcal{Z} \mapsto 2^{\mathcal{B}}$	Behaviours(z)表示在状态z中执行的行为集
Success：$\mathcal{P} \mapsto 2^{\mathcal{Z}}$	Success(p)表示规划p的终止状态集，表明规划的成功执行
Fail：$\mathcal{P} \mapsto 2^{\mathcal{Z}}$	Fail(p)表示规划p的终止状态集，表明规划没有成功执行
Post：$\mathcal{Z} \mapsto \mathcal{L}_S$	Post(z)为部分函数，将规划的终止状态与后置条件相对应
$\mathcal{U}$：$\mathcal{P} \mapsto 2^{\mathcal{L}_S} \mapsto \mathbb{R}$	$\mathcal{U}(p)$是p的效用函数，相对于一组公式，对p评估
Pref：$\mathcal{R} \times \mathcal{T} \mapsto [-1, 1]$	Pref(r, τ)是任务τ对角色r的优先权

第5章　语　　义

在前面的章节中，已介绍了 pALICA 语言及其语法，由此可区分语法错误和格式规范的程序。在此，将介绍格式规范的程序意义。

这里定义的语义有三个目的：首先，可直观理解 pALICA 程序，使得一个开发人员可轻松地设计多智能体程序；其次，详细规范了智能体如何执行程序，并定义了一系列可能的执行引擎；第三，可允许各种性能的机械证明，如存活性或安全条件。在上述三个目的中，认为第二个目的最为重要，这是因为我们更关心实际应用，即程序的执行。

接下来，声明了一组语义应遵循的规则。之后，通过一个以最普通方式定义的智能体模型，来介绍智能体如何理解 ALICA 的语义。然后，给定一个特定的 ALICA 程序，定义描述智能体的内部表示如何随时间变化的转移系统[122]的操作语义。

5.1　基本原则

ALICA 语义的设计遵循以下原则：域的独立性、自治性和局部性。这意味着在不同域特点下（如不可靠通信），可保证可移植性、扩展性和鲁棒性。

域的独立性　尽管机器人场景是本书的重点，但 ALICA 的设计具有域的独立性，使得本书结果可用于其他域，而并非局限于所讨论的域。因此，对于智能体如何表示其环境，ALICA 尽可能少地进行假设。例如，对于环境，完全可能利用一个封闭的环境假设或一个开放的环境假设，而且还可采用一个经典的一阶逻辑、一个模态逻辑或甚至一个如概率逻辑的混合方法。一个特定的 ALICA 程序可通过定义为签名的一部分的$\mathcal{L}$语言来表示环境。

自治性　在第1章中列举的要求之一是具有处理不可靠通信的能力。ALICA 可通过执行冗余计算来解决该问题，即由所有智能体独立自主地进行团队相关决策。随后，对不一致的决策进行检测，并根据相应信息（如通过通信或行为识别）进行修正。该原则使得 ALICA 即使在非常恶劣的网络条件下也可工作，如参考文献［159］所示。由于冗余计算，消息传递时间几乎无限制。对于该规则，只有一个例外，即有关无行为能力智能体的识别。在本书的实现中，假设如果在某一段时间内没有从一个智能体处接收到消息，则该智能体将不能再正常运行。所间隔的时间取决于网络质量、问题域的动态程度以及智能体失控的可能

性。在7.5节中，将对此进行详细讨论。

局部性 执行一个ALICA程序的团队智能体的全局状态由所有参与智能体的联合状态表示。为应对团队所面临问题的复杂性，ALICA采用层次化编程结构。所需求解的问题解由规划描述，即根据任务来分解团队，并采用子规划来求解可能的子问题。

保持树结构，并对所有决策执行必要计算可很好地解决单个智能体计算能力不足的问题，这或许是为什么首选多智能体系统的原因。因此，ALICA采用一种局部性原则。每个智能体按照其采用的规划，并根据这些规划执行相关决策。对于没有参与的团队的部分全局状态，该智能体可忽略。

如果问题可分解为子问题，则这种原则可显著减少大型团队的计算代价。因此相对于所参与智能体的个数，局部性原则大大增大了可扩展性，这也是人类处理复杂问题的本性。例如，在足球比赛中，若某个进攻队员正带球朝向对方球门，这时将无需考虑防守本方球门的队友的精确位置，只需知道有队友在防守即可。这等效于ALICA中在规划层次结构的较高层上已知对哪个任务分配了哪些智能体的思想。在动态程度较小的场景中，人类也采用同样的原则。如果你的同事参与大型合作项目，则无需考虑其所做的具体工作。实际上，这足以获悉一个抽象的进展情况。

5.2 智能体模型

在2.1节中，介绍了智能体的概念。根据所介绍的一个智能体的最通用模型，不作任何限制，只是在恰当位置增加ALICA控制结构。假设一个控制回路由信念更新、推理和执行三部分组成，ALICA将其集成到推理过程。

之前的信念更新集成了智能体对环境内部模型中的感知信息。在一个ALICA程序中，$\mathcal{L}$(Pred，Func）用于表示该模型中的实际情况和关系。推理步骤包括任何形式的内部计算，如规划和意图的更新。最重要的是，在该过程中更新了所执行ALICA程序的内部状态。也就是说，在此执行ALICA的计算步骤。

计算阶段按照推理过程所决定的动作命令来执行。在ALICA中，这是指发布动作命令的不可分行为程序。这些命令可进一步由中间推理步骤处理。最后，在该过程中还包括所有主动通信，这是由于认为发送信息的动作是其本身的一种行为。

并不要求智能体必须严格按照图2.1中的模型，这只是一个必要步骤的参考，以及所建议的步骤顺序。尤其是对于目前的并行硬件架构，需要采用更加模块化的架构。然而仍需将感知信息集成到内部模型，在推理步骤中反映结果，并

更新必要动作。在第7章，将介绍一种更精细的软件体系结构。

从 ALICA 的角度来看，一个智能体在任意时间点都有一个配置。该配置反映了智能体的内部状态、信念基的组成、角色以及执行 ALICA 程序中的状态。

定义 5.1（智能体配置） 对于任一智能体 $a \in \mathcal{A}$，用 $\mathrm{Conf}(a)$ 表示其配置。pALICA 中一个智能体的配置为一个元组(B, Υ, E, R)，其中

- B 为智能体的信念基。
- Υ为智能体的规划基。
- $E \subseteq \mathcal{B} \times \mathcal{Z}$是智能体的执行集，即执行智能体的行为—状态元组集。
- R 为当前角色 a 的集合。

信念基包括环境的当前模型，即智能体认为为真的任何事物。规划基是对程序中智能体当前状态的一种表示。执行集包含智能体所执行的行为 b 及其发生时的状态 z 所构成的所有元组(b, z)，称为 b 的文义。R 中包括团队中所有的角色 a。

最初，除了特定域的信念外，一个智能体的配置为空。因此启动后，智能体无法对团队的当前状态进行任何假设。特别是，智能体不相信应执行任何规划。

定义 5.2 一个智能体初始配置为$(B, \emptyset, \emptyset, \emptyset)$。这样使得 B 不包含有关 ALICA 程序中状态的任何信念。

操作语义定义了随着智能体执行程序如何更新配置。在介绍相应规则之前，首先将更加详细地讨论智能体配置的元素。

5.2.1 规划基

智能体的规划基反映了其当前的意图状态。具体而言，即表示针对所涉及的每个规划以及所执行的任务，智能体所处的状态。因此，每个元素都具有程序表示的意图。

定义 5.3（规划基） 一个智能体的规划基是一组由规划 p、任务 τ 和状态 z 组成的三元组(p, τ, z)。智能体 a 的规划基由 $\mathrm{PBase}(a)$ 表示。

如果(p, τ, z) 是对于智能体 a 的 $\mathrm{PBase}(a)$ 中的一个元素，则称参与 p（或执行 p）的 a 应执行任务 τ 并处于状态 z。一个智能体不能随意参与规划，而只能在某一时刻执行规划中的一个任务，同时只能执行规划类型中的一个规划。另外，智能体只能执行相应规划中的任务，并处于该任务内的可达状态。这种限制称为规划基公理。

定义 5.4 规划基公理集 Σ_p包含以下内容：

$$(p, \tau, z) \in \mathrm{PBase}(a) \wedge (p, \tau', z') \in \mathrm{PBase}(a) \rightarrow \tau = \tau' \wedge z = z' \tag{5.1}$$

$$(p, \tau, z) \in \mathrm{PBase}(a) \wedge (p', \tau', z') \in \mathrm{PBase}(a) \wedge p \in P \wedge p' \in P \rightarrow p = p' \tag{5.2}$$

$$(p, \tau, z) \in \mathrm{PBase}(a) \rightarrow \tau \in \mathrm{Tasks}(p) \wedge z \in \mathrm{States}(p) \wedge z \in \mathrm{Reachable}(p, \tau) \tag{5.3}$$

在此，定义以下反映规划基层次结构的宏：

定义 5.5 $\text{Plans}^+(\Upsilon, z)$ 表示智能体在状态 z 下规划基为Υ时所执行的规划集。可定义为最小集，使得：

- $p \in \text{Plans}^+(\Upsilon, z) \leftarrow (p, \tau', z') \in \Upsilon \wedge p \in \text{Plans}(z)$
- $p \in \text{Plans}^+(\Upsilon, z) \leftarrow p' \in \text{Plans}^+(\Upsilon, z) \wedge (p', \tau', z') \in \Upsilon \wedge p \in \text{Plans}(z') \wedge (p, \tau, z'') \in \Upsilon$

在实现层面上,规划基可采用一个结构而非集合来表示。利用一种图结构可更快捷地执行规则集。根据规则，单个实例的复杂度为 $O(1)$ 或 $O(n)$，而非 $O(n^2)$，其中 n 为规划基中三元组的个数。在 5.14 节中，将会表明若 pALICA 程序结构良好，则规划基总是可构成一个树结构，该特性进一步简化了程序的运行时表示。在此，假设采用一种可更直观描述语义而不侧重于效率的集合表示。在第 7 章，将详细讨论其具体实现。

5.2.2 信念基

信念基表示一个智能体关于环境的内部模型，其中包括对于执行同一规划的协调合作智能体的信念。在 ALICA 中，利用条件和效用以$\mathcal{L}(\text{Pred}, \text{Func})$ 形式进行表示。由此，ALICA 应对有关团队成员的信念表示进行某些假设。

设$\mathcal{L}(\text{Pred}, \text{Func})$ 为信念基的语言。$\mathcal{L}$为对于$\mathcal{A}$中每个智能体 a，由模态算子 Bel_a和 K_a扩展的一阶逻辑 ㊀。

算子 Bel_a表示智能体 a 的个体信念。$\text{Bel}_a\phi$ 表示智能体 a 信念为 ϕ。根据 Fagin等人[42]提出的知识公理，Bel 可由以下公理定义（如 KD45 系统）：

- $\text{Bel}_a(\phi \rightarrow \psi) \rightarrow (\text{Bel}_a\phi \rightarrow \text{Bel}_a\psi)$（分配公理）
- $\text{Bel}_a\phi \rightarrow \neg\text{Bel}_a\neg\phi$（一致性公理）
- $\text{Bel}_a\phi \rightarrow \text{Bel}_a\text{Bel}_a\phi$（肯定自省公理）
- $\neg\text{Bel}_a\phi \rightarrow \text{Bel}_a\neg\text{Bel}_a\phi$（否定自省公理）
- $(\forall x)\text{Bel}_a\phi(x) \rightarrow \text{Bel}_a((\forall x)\phi(x))$（知识量词）

这些公理形成强理性信念的概念。模态 K 将 Bel 向知识扩展：$K_a\phi \stackrel{\text{def}}{=} (\text{Bel}_a\phi) \wedge \phi$。

在个体信念的基础上，利用了“人人信念”（EBel）和共同信念（MBel）的常用概念。人人信念由下式定义（引自 Rao 等人相关研究[130]）：

$$\text{EBel}_A\phi \stackrel{\text{def}}{=} (\forall a \in A)\ \text{Bel}_a\phi$$

当且仅当 A 中所有智能体 a 的信念均为 ϕ，才满足公式 $\text{EBel}_A\phi$。ϕ 的共同信念定义为 A 中所有智能体 a 的信念均为 ϕ 且共同信念。$\text{MBel}_A\phi$ 可正式定义为

㊀ 原则上，由于条款（ground terms）集有限，ALICA 还可工作于命题式模态逻辑。

$$\mathrm{MBel}_A\phi = \mathrm{EBel}_A\phi \wedge \mathrm{EBel}_A\mathrm{MBel}_A\phi$$

的最大固定点。

由此可得到一个无穷组合形式：

$$\mathrm{MBel}_A\phi \leftrightarrow \mathrm{EBel}_A\phi \wedge \mathrm{EBel}_A\mathrm{EBel}_A\phi \wedge \cdots \wedge \mathrm{EBel}_A\cdots\mathrm{EBel}_A\phi \wedge \cdots$$

用文字可表述为，人人信念 ϕ 信任人人信念 ϕ 信任人人信念信任人人信念 ϕ 等。

$\mathcal{L}$(Pred, Func) 中包括 ALICA 的某些特定项，如对于 A 中每个智能体的常数，以及特定域的项，如坐标或表示实际物体的项。$\mathcal{L}$(Pred, Func) 中的各项由下式正式表示：

● 一个可数无穷的变量集，$X = \{x_1, x_2, \cdots, x_n, \cdots\}$。

● 一个 n 元函数符号（$n \geqslant 0$）Func，0 元函数符号称为常数。Func 包括：

➢ $\mathcal{A}$, $\mathcal{B}$, $\mathcal{Z}$, $\mathcal{P}$, $\mathcal{T}$, $\mathcal{R}$；

➢ 函数符号的特定域集合，F_{dom}。

为将程序执行中其他智能体的状态反映在信念基中，需引入下列谓词 (predicate)：

● In(a, p, τ, z)，当且仅当(p, τ, z) $\in$ PBase(a) 时满足。这使得智能体可推理对其他智能体内部状态的信念。例如，$\mathrm{Bel}_a\mathrm{In}(b, p, \tau, z)$ 表示 a 认为 b 将执行规划 p 中的任务 τ 以及 b 正处于状态 z。根据规划基公理 Σ_p，具有信念基的 In (a, p, τ, z) 与规划基中相应的规划—任务—状态三元组以同样方式受限。

● HasRole(a, r)，表示智能体 a 具有角色 r。

● Succeeded(a, p, τ)，当且仅当智能体 a 成功完成规划 p 中的任务 τ 时为真。

对于$\mathcal{L}$(Pred, Func) 的谓词集合，假设 Pred 已包含这些谓词。另外，由于基于规则的 ALICA 程序执行需要额外的状态信息，则假设一组额外谓词 P_{rules} 为 Pred 的子集。5.13 节中介绍的操作语义假设具有 P_{rules} 中的下列谓词：

● $\mathrm{Handle}_f(b, z)$，智能体可处理状态 z 中行为 b 的失败时满足。

● $\mathrm{Handle}_f(p)$，智能体可处理规划 p 的失败时则为真。

● Failed(p, i)，表示规划 p 失败 i 次。

● Failed(b, z, i)，表示行为 b 在状态 z 失败 i 次。

● Alloc(z)，当且仅当对于状态 z 下的智能体分配任务必要时为真。

● Success(b) 和 Failed(b) 表示行为 b 的成功或失败标志，如 4.1 节中所述。

最后，P_{dom}中包含有关智能体环境表示的谓词，如 DistanceTo(object, dist) 或 Carries(agent, object)。在此引入的语言元素使得智能体可推理环境（利用 F_{dom}和 P_{dom}中的符号）以及自身和彼此的内部状态。这样相对于智能体的信念，可对角色分配等进行计算。

为反映描述有关智能体内部状态信念的不同谓词之间的关系，在此定义了一组公理 Σ_b。

定义 5.6　设对于$\mathcal{A}$中智能体 a，Σ_b中包含下列公理：

● 智能体、行为、规划、状态、任务和角色的唯一命名公理：UNA（$\mathcal{A}$, $\mathcal{B}$, $\mathcal{P}$, $\mathcal{Z}$, $\mathcal{T}$, $\mathcal{R}$）

● 如果需要对规划失败进行处理，则

$(\mathrm{Bel}_a\mathrm{Handle}_f(p) \vee \mathrm{Failed}(p, i)) \rightarrow (p = p_0 \vee (\exists p', \tau, z)\mathrm{In}(a, p', \tau, z) \wedge p \in \mathrm{Plans}(z))$

● 同理，如果需要对行为失败进行处理，则

$(\mathrm{Bel}_a\mathrm{Handle}_f(b, z) \vee \mathrm{Failed}(b, z, i)) \rightarrow (\exists p, \tau)\mathrm{In}(a, p, \tau, z)$

● 只要有另一个智能体仍处于相应规划的状态中，则智能体成功完成任务：

$\mathrm{Succeeded}(a, p, \tau) \rightarrow (\exists z) p \in \mathrm{Plans}(z) \wedge (\exists a', \tau', p')\mathrm{In}(a', p', \tau', z)$

● 只需对智能体所处状态进行任务分配，则

$$(\mathrm{Bel}_a\mathrm{Alloc}(z)) \rightarrow (\exists p, \tau)\mathrm{In}(a, p, \tau, z)$$

定义 5.7（共同知识）　设 Σ_B为由以下给定的集合：

$$\Sigma_B = \Sigma_{\mathrm{syn}} \cup \Sigma_{\mathrm{dom}} \cup \Sigma_b \cup \Sigma_p$$

式中　Σ_{dom}——一组描述域的特定域公理；

Σ_b——根据定义 5.6 给出的一组信念基公理；

Σ_{syn}——语法约束集（见 4.5 节）；

Σ_p——根据定义 5.4 的规划基公理集。

Σ_B假设为$\mathcal{A}$中的共同知识，即满足 $\Sigma_B \wedge \mathrm{MBel}_{\mathcal{A}}\Sigma_B$。

定义 5.8（信念基）　一组公式 $B \subset \mathcal{L}$（Pred，Func）是针对智能体的一个信念基，当且仅当

$$\Sigma_B \cup B \mid \neq \bot$$

以及

$$\mathrm{In}(a, p, \tau, z) \in B \leftrightarrow (p, \tau, z) \in \mathrm{PBase}(a)$$

这样，一个智能体的信念基反映了对环境的信念以及对所有其他智能体内部状态的信念（即规划基）。上述定义产生一个反映智能体总是相信其意图所做行为的信念基。另外，相对于 Σ_B，信念基总是保持一致。用 BelBase(a) 表示智能体 a 的信念基。

定义 5.9（智能体证明）　设 a 为一个智能体，$\mathcal{F}$为$\mathcal{L}$中的一组公式，η 为一个替换，ϕ 为一个公式。相对于$\mathcal{F}$和 Σ_B，用下式表示 a 证明了 ϕ：

$$\mathcal{F} \vdash^{a}_{\mathcal{L},\eta} \phi$$

式中　η——证明的计算答案。

如果内容清楚，则可忽略 a、$\mathcal{L}$或 η。

证明算子$\vdash$是指逻辑$\mathcal{L}$中定理证明算法，且相对于域可互换。

5.2.3 信念更新

智能体的信念基不断更新，以适应新的感知数据、通信行为或内部状态更新。在此，仅分析最后一种情况。记 $B+F$ 表示信念基 B 由 $F=\{f_1, f_2, \cdots, f_n\}$ 的（有限）集进行更新：

$$B+F \overset{\text{def}}{=} B+\bigwedge_{f\in F} f$$

$$B-F \overset{\text{def}}{=} B+\bigwedge_{f\in F} \neg f$$

要求信念更新算子 + 满足 KM 假设[83] U1 ~ U4 和 U8，以适用于静态共同知识 Σ_B，从而 + 构成一个基本的惯性更新算子（参见 Lang 相关研究[92]）：

$$\Sigma_B \cup (B+f) \mid\ = f \tag{U1}$$

$$\Sigma_B \cup B \mid\ = f \rightarrow (B+f) \leftrightarrow B \tag{U2}$$

如果 $\Sigma_B \cup B$ 和 f 均满足，则 $\Sigma_B \cup\ (B+f)$ 也满足 (U3)

如果 $B \leftrightarrow C$，以及 $f \leftrightarrow g$，则 $B+f \leftrightarrow C+g$ (U4)

$$(B\cup C)+f \leftrightarrow (B+f)\cup(C+f) \tag{U8}$$

最重要的是，信念基必须在任何时候都一致。注意到，相对于 Σ_b，可通过去除 $\text{Handle}_f(p, z)$、$\text{Handle}_f(b)$、Failed(p, i)、Succeeded(a, p, τ)、Alloc(z)、Success(b, z) 和 Fail(b, z) 来达到一致性。

例如，假设一个智能体 a 执行一个规划 p，因此 In(a, p, τ, z) 在 BelBase(a)中满足。现在，由于更高层规划的作用，智能体放弃执行 p。则更新 BelBase(a)：BelBase$(a)'$ = BelBase(a) − In(a, p, τ, z)。例如，如果对于状态 z，所有分配都有待进行，Alloc$(z) \in$ BelBase(a)，则不再相关，因此 Alloc(z) 也会被去除。

5.2.4 执行集

智能体的执行集 E 满足(b, z) 元组，其中 b 为行为，z 为状态，使得 b 在 z 中发生（$b \in$ Behaviours(z)）。将状态 z 称为 b 的文义。可将 E 传递到执行行为底层的软件组件。E 中所包含的内容允许行为查询具有相应规划、状态或任务的智能体的信念基。操作语义必须保证如果(b, z) 在 a 的执行集中发生，存在规划 p 和任务 τ 使得(p, τ, z) 是其规划基的一个元素，从而使得 a 信念 In(a, p, τ, z)。5.14 节将阐述该特性。

5.2.5 角色集

一个智能体的角色集 $R\subseteq\mathcal{R}$是由智能体当前履行的所有角色组成的，由此满足 $\text{Bel}_a\text{HasRole}(a, r) \leftrightarrow r\in R$。角色由角色分配算法分配，并给出智能体执行任务的优先权。该两层方法简化了任务分配问题[21]。另外，利用角色和任务解耦智能体和规划，使得规划设定与团队组成相互独立进行，这通常在开发阶段是完

全未知的。通常，角色分配的全局最优解是 NEXP 时间完备的[104]。5.6 节中给出了一类简化问题的实际解决方案。

5.3 局部性

若允许规划中出现任意条件，则与 5.1 节中所定义局部性原则产生冲突。因此限制出现谓词 In(a, ρ, τ, z)，这样智能体就只能参考其自身的局部角度。

定义 5.10 当且仅当对于 ϕ 中所有的谓词 In(a, ρ, τ, z)，ρ 为恒定规划 p，则公式中 ϕ 对规划 p 而言是局部规划。

定义 5.11 局部性规定：所有作为规划的前置条件或运行时条件的公式，都是各自规划的局部规划。

该定义比需要强制执行的局部性原则有些严格。正如现在这样，在其他规划中完全禁止涉及智能体，可保证每个规划便于在其他情况下重用，这是一个对于规划算法以及非常复杂的 ALICA 程序的系统设计非常有用的特性。另外，正如将在 5.12 节中所述，该定义还允许规划的任务分配计算与其旁支和子规划均独立，由此极大简化了任务分配问题。

注意到，并不仅限于谓词 Succeeded(a, p, τ) 和 Failed(p, i)，也可用于其他规划。因此，规划可在特定情况下对子规划的成功终止或失败终止做出反应，另外转移条件也不受限。因此可在子规划中实现任意转移，从而产生了一些将在 10.1 节中讨论的建模模式。由于转移与求解任务分配问题不相关，因此不会产生任何问题。行为条件可分别处理（见 5.7 节）。

5.4 团队配置

在由谓词 In(a, ρ, τ, z) 在信念基中表示的执行规划的智能体概念基础上，引入执行规划的团队概念。在 ALICA 中，相对于所参与的规划，每个智能体不断监测其团队成员的动作。因此，一个智能体不仅可对智能体失控，并随后从团队中去除做出反应，而且每个智能体在决策时（如执行规划中的一个任务）还可考虑所有其他智能体的进展。注意到，由于局部性原则，对于没有参与的规划，智能体不能推断团队状态。

定义 5.12 设 a 为 A 中的一个智能体，TeamIn(A, p) 表示执行规划 p 的团队 $A \subseteq \mathcal{A}$，则可表示为

$$\text{TeamIn}(A,\ p) \overset{\text{def}}{=} (\forall \tau \in \text{Tasks}(p))(\exists n_1,\ n_2)\xi(p,\ \tau) = (n_1,\ n_2) \wedge$$

$$(\exists A')A' \subseteq A \wedge n_1 \leqslant |A'| \leqslant n_2 \wedge$$
$$(\forall a' \in A)a' \in A' \leftrightarrow (\exists z)\mathrm{In}(a', p, \tau, z) \vee \mathrm{Succeeded}(a', p, \tau)$$

如果 $A=\mathcal{A}$，可将 $\mathrm{TeamIn}(\mathcal{A}, p)$ 简写为 $\mathrm{TeamIn}(p)$。

定义5.12反映了执行规划的一个团队的最重要概念。在具有两个 A 和 A'，且 $A' \supset A$，使得 $\mathrm{TeamIn}(A, p)$ 和 $\neg\mathrm{TeamIn}(A', p)$ 的意义上，这是非单调的。因此，就智能体团队而言，智能体的信念对规划的评估具有重要作用。注意到，该定义还需考虑任务的成功完成。

5.5 成功语义

根据域的不同，某些行为和任务可以完成，即存在一个可实现的明确目标描述。一旦智能体达到目标描述要求，则认为已完成行为或任务，且该智能体可自由地去做其他工作。对于一个行为而言，可通过表示达到行为后置条件的谓词 $\mathrm{Success}(b)$ 来表明成功。一个完成规划 p 中任务 τ 的智能体 a 增强了 $\mathrm{Succeeded}(a, p, \tau)$ 的信念。如定义5.12中所示，任务成功完成会影响智能体如何考虑关于相应规划的团队状态。

在不同场景中，还应关注整个规划是否成功，而不是单个任务，然而智能体成功完成一个任务并不足以得出相应规划也成功的结论。实际上，规划的成功与否取决于任务的状态。由于规划的某些任务可选，因此需要一个新的定义来反映规划成功与任务成功之间的关系。

例如，考虑任务是将某个设备搬运通过未知区域的一组机器人。需要某些机器人在前方搜索来寻找最佳路线以避免搬运设备时经过不必要的路程。然而如果只有少数几个机器人，则为了都用于搬运机器人而忽略搜索环节。因此即使没有机器人完成搜索任务，规划仍是成功的。

定义5.13　对于一个规划，任务可能是可选的也可能是必须完成的。在此，将规划 p 所需的任务集记为 $\mathrm{Required}(p)$。若对于规划所需的所有任务，成功完成任务的智能体个数最少，则认为规划是成功的：

$$\mathrm{Succeeded}(p) \stackrel{\mathrm{def}}{=} (\forall \tau \in \mathrm{Required}(p))(\exists n, n', m)\ \xi(p, \tau)=(n, m)$$
$$\wedge n'=\max(1, n) \wedge n' \leqslant |\{a \mid \mathrm{Succeeded}(a, p, \tau)\}|$$

因此，对于规划所需的任何任务，能执行该任务的智能体也必须是完成该任务个数最少的智能体。在最小基数为零时，至少需要一个智能体来完成任务。

再次考虑上述搬运场景的例子。在智能体的搬运路径上可能存在废墟。通过利用零最小基数，可使得所有智能体先放下设备，在另一个任务中清除废墟，然后在不放弃规划执行的条件下继续搬运。

例如，如果规划—任务对(p, τ)的基数为（2，4），这意味着对于要完成的规划p，必须至少有两个智能体来执行任务τ，当然实际执行任务的智能体个数可以更多。尽管一开始必须至少有两个智能体来执行任务τ以满足TeamIn(p)，但对于完成任务的智能体，这并非必要条件。

在智能体正在完成一个规划时，并不需要一次完成所有任务。实际上，某些任务可能会较早完成，这时相应的智能体可自由地去做别的工作。给定一个信念集，函数ξ_t：$\mathcal{P} \times \mathcal{T} \times 2^{\mathcal{L}_S} \mapsto \mathcal{N}_0 \times (\mathcal{N}_0 \cup \{\infty\})$反映了当前所需的规划基数：

$$\xi_t(p, \tau, B) = \begin{cases} (n, m) & \tau \notin \text{Required}(p) \\ (\max(0, n-c), \max(0, m-c)) & \text{其他} \end{cases}$$

式中　$\xi_t(p, \tau, B) = (n, m)$；

$c = |\{a \mid \text{Succeeded}(a, p, \tau) \in B\}|$。

在5.8节讨论的任务分配中，利用了以ξ_t反映的所需智能体的概念。

根据信念公理Σ_b，直到所有智能体都完成相应规划，任务的成功才有效。如果智能体已完全完成一个规划，而又在后来的某段时间内重新执行该规划，则必须再次完成该规划，即认为第二次执行该规划是针对一个新的问题。否则一旦完成规划，则再也不能执行，直到通过某些其他方法重置智能体。

5.6 角色分配

在4.4节中，介绍了角色的基本概念。现在讨论一种为智能体分配角色的方法。Campell和Wu[20]对目前已有的角色分配算法进行了详细的综述。在此，并不讨论角色分配算法的研究现状，而是通过实例来阐述如何将现有方法集成到ALICA中。

类似于Vail和Veloso[170]，在此采用一种广播计算方法，其中智能体广播有关角色分配的信息，并对分配进行局部计算，由此得到较高反应性的分配。然而与上述方法相比，ALICA中的角色相对固定，这是由于其不依赖于域的高动态特性，如机器人位置或其他感兴趣的物体。实际上，这仅取决于智能体的个数和能力。任务分配所反映的更多动态特性，将在5.8节中讨论。

为给智能体分配角色，需要一个表示智能体对某一角色充分性的量度，即角色效用，该效用可分别由角色和智能体所需的和所提供的能力来确定。假设一个相似性度量Δ：$\mathcal{C} \times \mathcal{C} \mapsto [0, 1]$，使得$\Delta(x, x) = 1$，$\Delta(x, x^c) = 0$，其中$x^c$表示$x$的补。这种基于模糊集的量度可参见参考文献［186］。

定义5.14　（角色效用）对于某一角色r，某一智能体a的效用$\mathcal{U}(a, r)$是基于智能体的能力Cap(a)和角色所需能力Cap(r)之间的相似性Δ。这些值的归一化之和反映了智能体对于某一角色的效用：

$$
\mathcal{U}(a,r)=\begin{cases} 0 & (\exists c \in \mathrm{Cap}(r)) \\ & \max_{c' \in \mathrm{Cap}(a)} \Delta(c,c') = 0 \\ \frac{1}{|\mathrm{Cap}(r)|} \sum_{c \in \mathrm{Cap}(r)} \max_{c' \in \mathrm{Cap}(a)} \Delta(c,c') & \mathrm{Cap}(r) \neq 0 \\ 1 & \text{其他} \end{cases}
$$

也就是说，所需的每个能力都与智能体的能力最佳匹配。

给定角色效用，可定义一个对团队中所有智能体分配角色的角色分配过程。假设该过程由一个队形封装。

定义 5.15　对于一个给定的 ALICA 程序，用 F 表示队形。队形中包含特定的角色分配算法以及角色分配计算所需的任何信息。角色分配是由 $F(A)$ 计算的形式为 HasRole(a，r) 的一组信念，其中 A 为智能体集合。

一个简单有效的队形可由优先级列表描述。优先级列表定义了$\mathcal{R}$中所有角色实例的总排序，并根据总排序为最适合的智能体分配最重要的角色。表 5.1 给出了相应的算法。在这种特定情况下，如果智能体个数多于角色个数，则算法从其余智能体开始执行。另外还要注意，0 值用于表示智能体没有能力来担任某一特定角色，即表示该智能体缺乏特定能力。

表 5.1　基于优先级的角色分配

```
A := set of available agents;
R := non—empty sorted list of roles according to their priority;
i := 0;
j := 0;

while (A ≠ Ø) {
  a := argmax_{a∈A} U(a, r[i]);
  if (U(a, r[i]) > 0) {
    assign a to role r[i];
    remove a from A;
    j := 0;
  } else j++;
  i := (i+1) mod |R|;
  if (j = |R|) return FAILURE;
}
```

在机器人足球领域，角色实例的一个合适列表可能为

$$\vec{R} = (\text{Attacker}, \text{Goalie}, \text{Defender}, \text{Attacker}, \text{Defender}, \text{Supporter})$$

这描述了首先需要一个进攻机器人，然后是守门员和防守机器人。如果还有另外的机器人，再分配第二个前锋和第二个后卫，然后是中场。这时，能力是表示只

有尺寸较大的机器人可担任守门员角色，而能踢球的机器人作为前锋。如果由于丧失能力而导致一个机器人不能分配角色，则在对所有可担任角色的机器人分配之后，算法也是失败的。

最先进的技术可处理每个角色的基数，并考虑相互关联的效用。但由于该问题与5.8节所讨论的任务分配问题类似，故在此不予讨论。值得注意的是角色分配仅依赖于智能体的个数和能力，因此仅只需计算上述发生变化的事件，如智能体丧失能力或团队中新增一个智能体。

该方法集成了可检测智能体或机器人故障组件的监测设备。这种情况下，角色分配可对刚丧失的或退化的能力作出反应，并为损坏的机器人分配一个不太重要的角色。Weber 和 Wotawa[177] 提出了一种类似方法。在5.13节中提供了一种将角色分配集成到 ALICA 运行时语义的操作规则。

5.7 正则行为规划

行为是所构 ALICA 程序中的核心基元（原语）。协调和合作依赖于哪个智能体执行何种行为的准确信息，以及该信息是显式表示还是隐式表示。在 ALICA 中，行为封装了不可分的动作程序，将其看作一个由前置条件、后置条件和运行时条件注释的黑盒。与规划不同，在此没有引入反映智能体是否执行行为的信念，由此智能体就不能直接推理另一个智能体所执行的行为，但是智能体可推理相应的更高层结构的规划。当行为由只有一个局部智能体评估的关键前置条件和运行时条件注释时，这种不同显而易见。

通过引入表示行为执行的正确信念可缩小之间的差别，然而这种信念会使得操作语义不必要的复杂化，由此导致可读性变差。然而值得注意的是，行为可嵌入到反映行为条件的规划中，因此可挂起行为使得团队可推理和交流个体智能体和特定行为之间的关系。

这种嵌入必须保证每当规划执行时，同时也执行行为。通过对每个行为均产生一个规划和一种规划类型，可自动实现上述嵌入。具有包含行为和终止状态的两种状态的规划可实现一个简单的嵌入，在此将这种简单嵌入称为行为的一个正则规划。

定义5.16（正则规划） 行为 b 的一个正则规划 p_b 由下列结构定义：

- $\text{States}(p_b) = \{z_b, z_b^s\}$。
- $\text{Tasks}((p_b) = \{\tau_0\}$。
- $\text{Init}(p_b, \tau_0) = z_b$。
- $\xi(p_b, \tau_0) = (0, \infty)$。
- $\text{Pre}(p_b) = \text{Pre}(b)$，$\text{Run}(p_b) = \text{Run}(b) \wedge \neg \text{Fail}(b)$。

- $\mathcal{U}(p_b)(B)=1$。
- $\text{PlanTypes}(z_b)=\emptyset$。
- $\text{Behaviours}(z_b)=\{b\}$。
- $\mathcal{W}\ni(z_b, z_b^s, \text{Succeeded}(b))$。
- $\text{Success}(p_b)=\{z_b^s\}$，$\text{Fail}(p_b)=\emptyset$。
- $\text{Post}(z_b)=\text{Post}(b)$。
- $\text{Required}(p_b)=\{\tau_0\}$。

正则规划是相应正则规划类型的单一元素：

定义5.17（正则规划类型） 行为 b 的一个正则规划类型 P_b 正好包含 b 的正则规划，$P_b=\{p_b\}$。

假设每个行为都嵌入在一个正则规划中，就单独考虑行为条件而言，智能体是完全自由的，这就足以考虑规划条件。在可能需要采用实现附加结构的其他嵌入时，这将不会产生额外的表现力。因此，接下来假设每个行为只在其正则规划中发生。

5.8 任务分配

多智能体协调合作的一个最核心的问题就是任务分配[20,104]。在ALICA中，该问题是由从规划类型中选择一个规划并在所选的规划中对智能体分配任务定义的。由于ALICA允许每个任务可具有多个智能体，因此该问题是NP难题[55]。与Nair等人[104]提出的任务分配算法不同，可能涉及的每个规划的效用函数取决于完全分配，这样在完全分配未知的条件下，就无法计算每个智能体的效用函数。

每当智能体进入一个状态，就会产生任务分配问题。此时，必须对该状态下所有规划类型进行决策，应执行哪个规划，以及如果可能应执行哪个任务。也就是说，必须对所有参与的智能体进行任务分配的计算。按照保证快速反应的广播—计算方法，智能体根据其当前信念来计算任务分配。理想情况下，每个参与的智能体都计算同一任务分配，然而这不能完全保证，因为参与的智能体可能具有影响计算结果的不同感知数据。ALICA执行弱承诺思想。假设不断修正所计算的分配，直到产生相互矛盾的信息。第6章将主要讨论如何修复分配以及冲突消解。

对于规划 p，其任务分配 C 是一组谓词 $\text{In}(a_i, p, \tau, z)$，对 p 中积极参与的每个智能体 a_i 进行分配。对于任务分配 C 中的每个 $\text{In}(a_i, p, \tau, z)$，$\tau$ 为与 p 相关的任务，即 $\text{Tasks}(p)$ 中的一个元素，z 是 p 中 τ 的初始状态 $\text{Init}(p, \tau)$。在没有一个参考框架下，无法进行任务分配计算。通常将问题域的情况作为参考框

架，如当前情况或规划假设的情况。在此，将该参考框架表示为$\mathcal{L}$的语法集。与信念更新的概念一样，这组假设有时也作为信念状态。值得注意的是，这些假设可能包含 $\mathrm{In}(a, p, \tau, z)$ 的含义，从而使得某些智能体已在 p 中分配某些任务。

定义 5.18 任务分配 C 是一组形式为 $\mathrm{In}(a, p, \tau, z)$ 的基本信念。

定义 5.19 在假设集合$\mathcal{F}$下，规划 p 的任务分配 C 有效，当且仅当：

- $(\forall l \in C)(\exists a, \tau) l = \mathrm{In}(a, p, \tau, \mathrm{Init}(p, \tau)) \wedge \tau \in \mathrm{Tasks}(p)$。
- $\Sigma_B \cup \mathcal{F} \cup C \mid \neq \perp$。
- $\Sigma_B \cup \mathcal{F} \cup C \vdash \mathrm{Pre}(p) \wedge \mathrm{Run}(p)$。
- $\Sigma_B \cup \mathcal{F} \cup C \vdash \mathrm{TeamIn}(p)$。
- $\mathcal{U}(p)(\mathcal{F} \cup C) \geqslant 0$。

当且仅当在假设$\mathcal{F}$下，C 为规划 p 的一个有效任务分配，宏 $\mathrm{TAlloc}(p \mid \mathcal{F})(C)$ 的定义才满足。

根据该定义，一个有效任务分配应符合 ALICA 公理以及相应情况$\mathcal{F}$。如果在当前状态下分配智能体，$\mathcal{F}$就相当于智能体的信念基。因此，应首先考虑已执行 p 中一个任务的智能体，这样可允许在某些智能体退出规划且需要替换的情况下进行动态修复。另外，该定义还保证所分配的规划能够由智能体团队正确执行。最后，由于需要一个正效用，效用函数可禁止在特定条件下执行某些分配任务。

一个规划的效用函数不仅表明了规划符合某个特定情况的程度，而且还应考虑任务分配。因此，效用函数可反映与 Nair 等人[104]所提方法类似的对于某些特定任务的奖励。对于一个分配了规划 p 中任务的智能体应最大化 $\mathcal{U}(p)(\mathcal{F} \cup C)$，其中 C 是一个有效任务分配，$\mathcal{F}$为如当前信念情况的假设。换句话说，即计算

$$T = \operatorname*{argmax}_{C} \mathcal{U}(p)(\mathcal{F} \cup C)$$

在 $\mathrm{TAlloc}(p \mid \mathcal{F})(C)$ 条件下。

定义 5.20 在假设集合$\mathcal{F}$下，对规划类型 P 而言，一个任务分配 C 有效，当且仅当 P 中存在一个使得 $\mathrm{TAlloc}(p \mid \mathcal{F})(C)$ 的规划 p。

根据定义 5.20，规划类型的一个有效任务分配也是该规划类型中所有规划的有效任务分配，从而可从规划类型中选择一个规划。这样就使得优化任务还包括规划 p 的选择：

$$T = \operatorname*{argmax}_{C} \max_{p \in P} \mathcal{U}(p)(\mathcal{F} \cup C)$$

在 $\mathrm{TAlloc}(p \mid \mathcal{F})(C)$ 条件下。

由于只考虑了该优化任务的有效分配，因此在所有情况下，可能不会存在解。

5.9 递归式任务分配

在任务分配概念的基础上，介绍一种将智能体分配给规划树中一个规划分支的递归式任务分配。直观上，只要智能体进入一个状态，就必须求解该状态内所有规划类型的任务分配问题，进入该分配所涉及的状态，然后还要解决这些子状态中的任务分配问题。

由于 ALICA 程序的层次化结构特性，递归式任务分配问题是 ALICA 的核心问题。在 5.13 节中，将讨论在具有环境动态变化或不可预见问题（如智能体损坏）的执行过程中，如何动态自适应地进行任务分配。这种任务动态重分配需要一种实时产生最优分配的高效任务分配算法。

下面的示例展示了递归式任务分配问题，并利用该示例来讨论分配算法的特性。

示例 5.1　一个大型餐厅经理希望使用机器人为顾客提供服务，为此购买了三个服务机器人，分别命名为 a、b 和 c。对这些机器人制定了一个顶层规划 P0，其余规划由同事完成。图 5.1 给出了 P0 及其同事制定的规划。为简单起见，每个规划类型仅包含一个规划。这样当执行ServeGuests 时，机器人既可从厨房取食物，并送到餐桌（DeliverOrder 任务），也可服务顾客，如果需要的话还要下单（TaskOrder 任务）。餐厅的厨师希望尽快将食物送出以免变凉，因此要求传递食物的机器人应尽可能多。将上述过程表示为一个效用函数，记为

$$\mathcal{U}(\text{ServeGuests}) = \frac{|\{a \mid \text{In}(a, \text{ServeGuests}, \text{DeliverOrder}, z\}|}{100} + 0.1\,\frac{|\{a \mid \text{In}(a, \text{ServeGuests}, \text{TakeOrder}, z)\}|}{100}$$

式中　100——服务于该餐厅的所有机器人的最大绝对值。

聪明的手下制定了一个为餐桌传菜的规划 DeliverOrder。已知机器人在等待下一步任务时只是在餐桌周围闲逛，并会占用空间，因此增加一个前置条件，即表示在厨房准备食物时，无需机器人执行 FetchOrder：

$$(\exists x) \mid \text{In}(a,\ \text{DeliverOrder},\ \text{Waiter},\ Z_3) \mid \leqslant x \wedge \text{DisherAvail}(x)$$

现在，考虑机器人刚开始工作，且有两道菜需要服务（DishesAvail（2））的场景。对规划 P0 中的状态 Z_0 执行递归式任务分配。最大化规划 ServeGuests 的效用，即规划类型 ServeGuestsPT 中唯一的一个规划，这将对任务 DeliverOrder 分配所有三个机器人，即需要整个团队来执行任务 DeliverOrder。然而由于只需服务两道菜，在任务 DeliverOrder 中分配三个机器人会与其前置条件产生冲突。因此，只能为 DeliverOrder 任务分配两个机器人，另一个机器人应服侍顾客，并

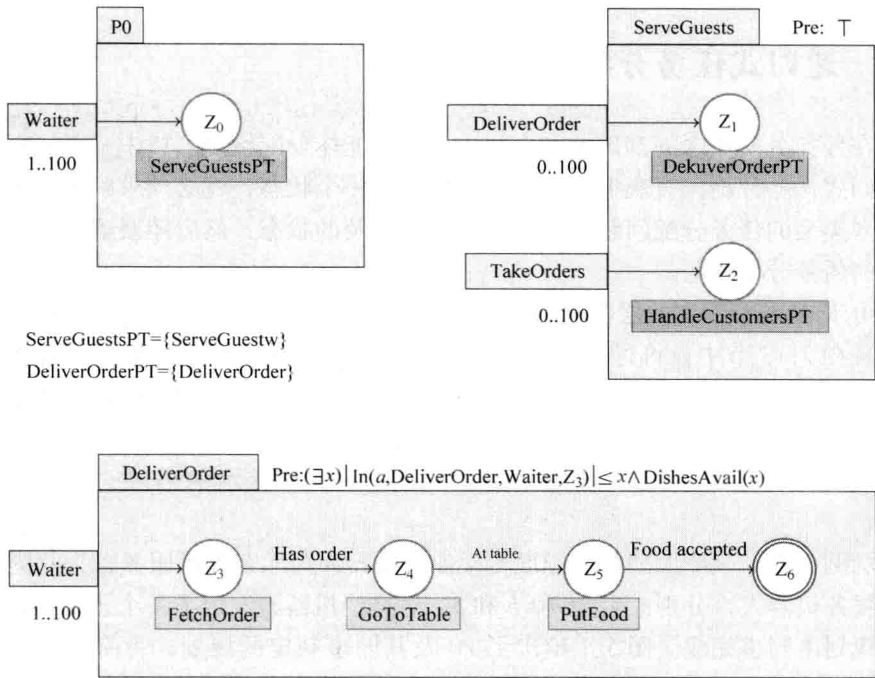


图 5.1 任务分配示例

等待下一步命令。

在上述简单实例中，对规划 ServeGuests 中的两个任务分配三个机器人会产生 8 种可能性。一般而言，在具有 n 个机器人，m 个任务的一个规划中，所有可能存在的分配个数为 m^n，这无法快速地对所有可能性进行强制穷举，因此需要一种更有效的方法。在 5.11 节中，将讨论如何通过构建效用函数来应用如 A* 算法等搜索算法。

然而，在单个规划中机器人的分配只是所研究问题的一部分。在该示例中，父规划 ServeGuests 与两个子规划的分配可相互作用。一个可计算满足所有条件并相对于所涉及效用函数最优解的算法应考虑所有参与规划的全局问题，这将抵消层次分解在计算时间上的优势。另外，由于每个智能体都需要计算分配，而通信延迟可能太大且通信太不可靠，使得多个智能体的复杂规划结构可有效克服单个机器人计算能力的不足。最后，未参与智能体无法获得规划 p 中分配决策所需的必要信息，因此认为全局分配算法不可行。

在 5.1 节中，引入局部性原则，即要求每个智能体只考虑其所参与的规划，而无需计算其余未参与的规划。根据这一原则，三个智能体可能分别得出结论：不能对任务 DeliverOrder 分配所有三个智能体。根据智能体和任务的总排序，每

个智能体又都能独立地得出结论：智能体 b 和 c 应负责传送食物，而智能体 a 负责下单。智能体 b 和 c 需要确定任务 DeliverOrder 的前置条件为真，而智能体 a 无需评估。图 5.2 表明了上述情况。其中，所有智能体对分配方案达成一致，并实现协调合作。但是如果只需传送一道菜呢？这时就只需一个智能体来执行任务 DeliverOrder，如图 5.3 所示。

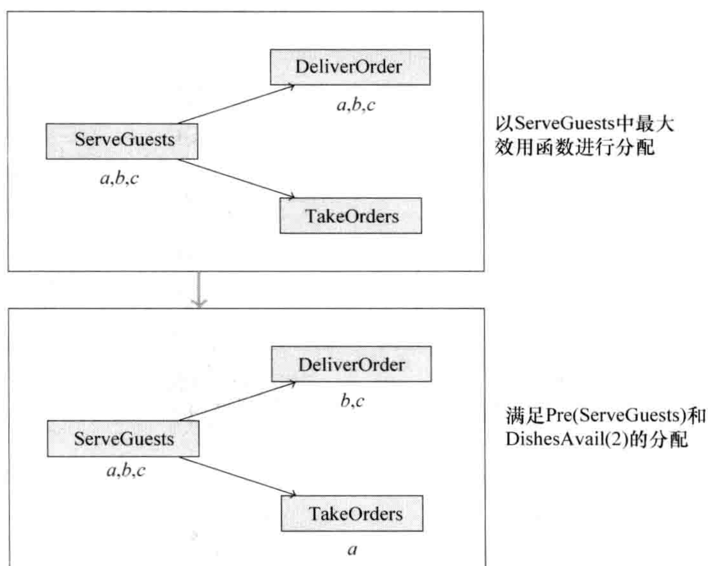


图 5.2 餐厅场景中的递归式任务分配

对于该修正信念，所参与的智能体对不同分配进行计算，会产生相互冲突的情况。相对于 ServeGuests 的效用函数，每个机器人都认为另一种分配方案为最佳有效的方案。乍看上去，这似乎是局部性原则的内在固有问题，因为相对于域，在不同层次上规划的条件和效用函数可相互作用，即通过$\mathcal{L}$中的任意关联。

这种效果可以通过下列方式抵消：

全局分配 一个全局分配算法不服从局部性原则，因此即使在上述情况下也会产生一致分配。然而随着智能体的个数、规划树的深度和广度的不断增大，全局分配计算很快就不可行。另外如上所述，任何一个智能体都无法获得全局分配计算所需的必要信息。

转化 在规划运行之前的计算步骤可通过集成子规划的相关信息来更改条件和效用函数，从而避免上述情况。然而该过程取决于$\mathcal{L}$，尤其是蕴涵背景知识方面的可判定性。对于一阶逻辑，这通常是不可判定的。如果 ALICA 程序是由一个规划算法产生的，则该算法需注意所产生的程序应避免上述的结构问题。接下来，将讨论规范格式所包含的内容。

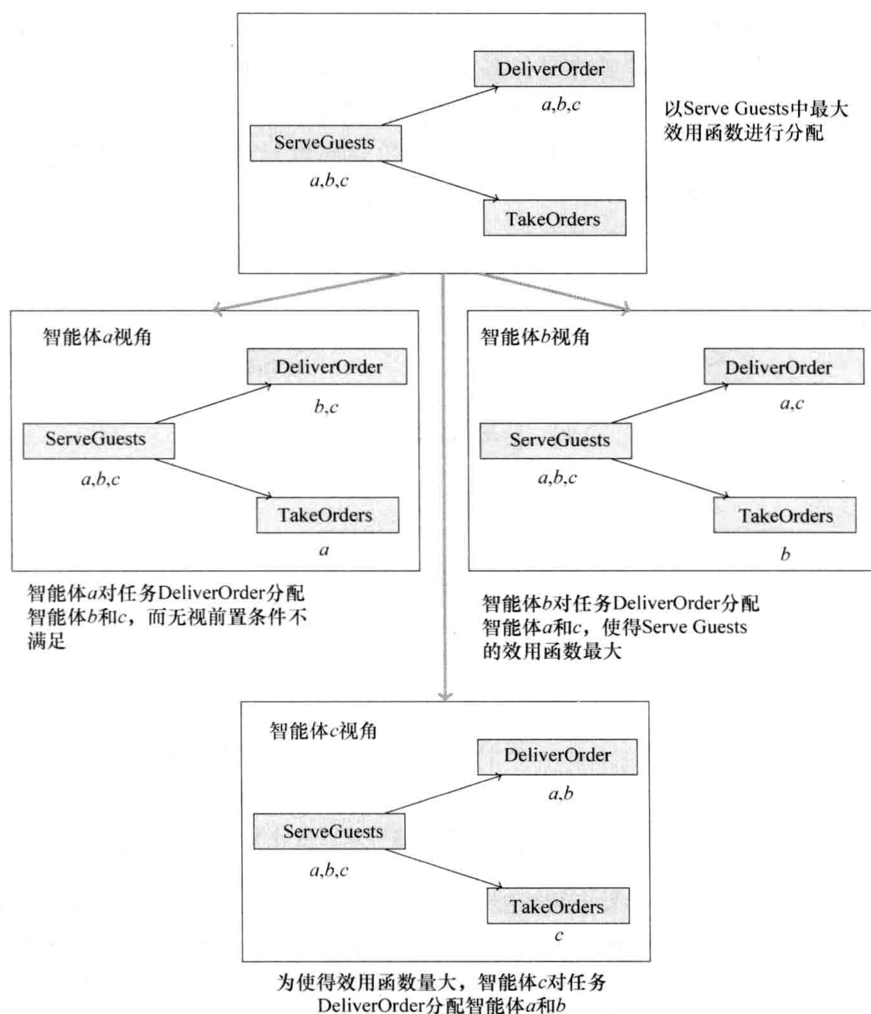


图 5.3　餐厅场景下不一致的任务分配

通信　智能体还可通过通信来实现一致性分配。实际应用中，通信过程占用大量时间且不可靠，但在某些场合或许是一个合理选择。例如，一个智能体可通知团队成员其无法参与某一规划，然后每个智能体将该信息纳入到分配过程中。然而在高动态领域，并不清楚该信息能有效多久。在极端情况下，这些信息或许在被所有智能体接收之前已无效，从而不可能实现一致性分配。由于本书主要考虑动态领域，因此应避免该选项。

另外，对于每个规划，智能体可选择其中一个作为领导者来计算局部分配，从而实现分配一致性。例如，在 STEAM[163,125] 中就是采用该方法。领导者的选举

过程会耗费一定时间，并可能引入其他问题：若所选举的领导者损坏，需重新选择另一个智能体作为新的领导者，这样就可认为整个团队通常应能在没有一个明确领导者情况下正常工作。在第6章，将讨论在团队合作中检测到冲突时如何触发选举过程，以及所产生的领导者如何负责某一具体规划类型的任务分配问题。

解耦 通过解耦分配问题可避免相互作用条件的问题，通过允许智能体不参与规划而实际上闲置来实现。从直观上看，这类似于对每个基数为(0，∞)的规划增加了一个任务，使得对于该任务条件和效用函数是中性的。在餐厅示例中，这将导致对任务 DeliverOrder 分配的所有智能体最大化 ServeGuests 的效用函数，而多余的智能体将空闲，在新菜出来之前不参与规划 DeliverOrder。

尽管解耦方法使得对于全局问题的分配并非最优，但仍具有一些显著性能。首先，该方法可保证在给定相对于每个规划的条件和效用函数智能体的信念基均等效的条件下，智能体对每个规划的分配达成一致。其次，该方法与转化方法和通信方法相兼容，即即使在分配问题已解耦的情况下，也可集成转化方法或通信方法。因此，将讨论规划分配和子规划分配之间不同程度的解耦，并在局部性原则下形成相应的概念。

目的是定义一种仅限定于这些规划的任务分配的递归式分配，使得所分配的智能体都依照该分配形式。按照这种方法，可实现通过层次化分解来减少计算复杂性的思想。对于包含智能体分配树形结构的状态，总是采用递归式任务分配。

首先，定义了一个分配必须满足的最低要求，然后是使得规划分配和子规划分配之间紧耦合的严格定义。

定义 5.21 （递归式有效任务分配）设 C 为在假设条件 $\mathcal{F}$ 下智能体 a 对状态 z 的一个任务分配。设 $\vec{P}$ 为 C 中所涉及的规划集，即 $\vec{P}=\{p \mid (\exists a', \tau, z')\mathrm{In}(a', p, \tau, z')\in C\}$，并将 $\mathcal{F}\cup C$ 记为 $\mathcal{G}$。则 C 为递归式有效，当且仅当：

$$\Sigma_B \cup \mathcal{G} \mid \neq \perp \quad (5.4)$$

$$\Sigma_B \cup \mathcal{G} \vdash \Big(\bigwedge_{p\in\vec{P}} \mathrm{Pre}(p) \wedge \mathrm{Run}(p) \Big) \quad (5.5)$$

$$(\forall p \in \vec{P})\mathcal{G} \vdash \mathrm{TeamIn}(p) \quad (5.6)$$

$$(\forall p \in \vec{P}) p \in \mathrm{Plans}^{+}(z) \wedge (p \in \mathrm{Plans}(z) \vee (\exists p', \tau', z') \mathrm{In}(a, p', \tau', z') \in \mathcal{G} \wedge p \in \mathrm{Plans}(z')) \quad (5.7)$$

$$(\forall l \in C)(\exists a', p, \tau) l \in \mathrm{In}(a', p, \tau, \mathrm{Init}(p, \tau)) \quad (5.8)$$

$$(\forall p \in \vec{P})\mathcal{U}(p)(\mathcal{G}) \geqslant 0 \quad (5.9)$$

$$(\forall a')(\exists p', \tau', z')\mathrm{In}(a', p', \tau', z') \in C \rightarrow \mathcal{F} \vdash (\exists p'', \tau'')\mathrm{In}(a', p'', \tau'', z) \quad (5.10)$$

设 $\mathrm{TAlloc}^{*}(a, z \mid \mathcal{F})(C)$ 表示假设条件 $\mathcal{F}$ 下，由智能体 a 对于状态 z 完成

的 C 为一个递归式有效任务分配。

根据式（5.4），一个递归式有效任务分配与假设 $\mathcal{F}$ 和共同知识相一致。因此，根据定义 5.3 和 5.8，一个智能体不能分配两个相互冲突的任务。尤其是，已确认参与规划 p 的智能体不能在该规划中再次分配任务。式（5.5）确保满足参与规划的所有前置条件和运行条件。式（5.6）排除了部分分配的规划，即确保每个规划都由适当数量的智能体执行。式（5.7）限定了规划类型所对应规划的任务分配，该规划类型是对应于通过采用分配，智能体所处的状态。

根据式（5.8），认为新分配任务的每个智能体均处于相应的初始状态，即不能分配到该任务的任意状态。式（5.9）保证了所有效用函数结果均不小于 0，由此认为这些分配是没有危害性的。最后，根据式（5.10），只能对处于状态 z 的智能体进行分配，认为这些分配已计算完成。

寻找该问题最优解的任务称为递归式任务分配问题，这将在后面详细讨论。

值得注意的是，空集也是一个递归式有效任务分配。另外，一个递归式有效任务分配并不需要保证执行所有参与的规划类型。这将允许忽略没有执行的规划，并且当团队中其余成员完成一个任务时，允许没有参与的那些智能体空闲。

另一个特性，即完备性，需要执行更严格的要求。

定义 5.22　在假设条件 $\mathcal{F}$ 下，智能体 a 对状态 z 完成的一个递归式任务分配 C 是递归完备的，当且仅当其递归有效且

$$(\forall P \in \mathrm{PlanTypes}(z))(\exists p \in P)(\mathcal{F} \cup C \vdash \mathrm{TeamIn}(p)) \wedge \\ (\forall \mathrm{In}(a, p', \tau, z') \in C)(\forall P \in \mathrm{PlanTypes}(z')) \\ ((\exists p'' \in P)(\mathcal{F} \cup C \vdash \mathrm{TeamIn}(p'')) \tag{5.11}$$

用 $\mathrm{TAlloc}_c^*(a, z \mid \mathcal{F})(C)$ 表示 C 为假设条件 $\mathcal{F}$ 下，智能体 a 对状态 z 完成的一个递归完备的任务分配。

一个递归完备的任务分配要求对于初始状态 z 以及局部智能体 a 所分配的每个状态，团队需执行所包含的每个规划类型中的一个规划。换句话说，如果在任何时候都满足该特性，则智能体必须保证对于所处的每个状态 z，以及 $P \in \mathrm{PlanTypes}(z)$ 的所有规划类型，都要执行一个 $p \in P$ 的规划。注意，智能体 a 并不需要必须参与所有规划的执行，只需确保整个团队要执行所有规划。这类似于联合意图[97]，即所有参与智能体都要执行规划，即使某些智能体并没有真正参与执行。当参与智能体损坏以及规划执行存在风险时，为完成规划执行，没有参与的智能体必须要积极参与。

强制实施完备性特性并不会对上述场景产生任何问题，但当一个智能体或一组智能体不能执行子规划，而这又是更高层所需的分配时，会导致团队状态产生冲突。

完美分配概念反映了单个分配问题的更加紧密的耦合：

定义 5.23　在假设条件$\mathcal{F}$下，智能体 a 对状态 z 完成的一个递归式任务分配 C 是递归完美的，当且仅当其递归有效且

$$(\forall P \in \mathrm{PlanTypes}(z))\mathrm{In}(a',p',\tau,z) \in \mathcal{F} \cup C \rightarrow$$
$$(\exists z',p'',\tau')p'' \in P \wedge \mathrm{In}(a',p'',\tau',z') \in \mathcal{F} \cup C \tag{5.12}$$
$$\mathrm{In}(a,p,\tau,z') \in C \rightarrow ((\forall a')\mathrm{In}(a',p,\tau,z') \in C \rightarrow$$
$$(\forall P \in \mathrm{PlanTypes}(z'))(\exists p',\tau',z'')p' \in P \wedge \mathrm{In}(a',p',\tau',z'')) \tag{5.13}$$

类似于之前定义，设 $\mathrm{TAlloc}_p^*(a, z \mid \mathcal{F})(C)$ 表示 C 为在假设条件$\mathcal{F}$下，智能体 a 对状态 z 完成的一个递归完美的任务分配。

式（5.12）和式（5.13）要求对所有智能体进行任务分配。即每个处于初始状态 z 的智能体均部分执行 z 中每个规划类型［见式（5.12）］。另外，根据式（5.13），对于递归式任务分配，也同样如此。如果分配智能体 a，使其处于某一特定状态，则在该状态下的所有智能体都必须参与执行该状态所包含的所有规划类型。

显而易见，根据定义 5.21 中的式（5.6），一个递归式完美分配也是递归式完备分配。完备分配和完美分配都会存在耦合问题，即在不同层次上以该方式相互作用的规划效用函数和条件，使得智能体不能单独计算各自分配。

在某种程度上，规划应具有一定的兼容性以避免这种相互作用。这种兼容概念取决于域公理，Σ_{dom}。

定义 5.24　规划类型 P 与规划 p 中的任务 τ 分层兼容，当且仅当对于所有的信念基$\mathcal{F}$，如果在假设条件$\mathcal{F}$下存在对于规划 p 的一个有效分配 C，同样在假设条件$\mathcal{F} \cup C$ 下也存在一个对于规划 $p' \in P$ 的一个有效任务分配 C'，使得$\{a \mid \mathrm{In}(a,p,\tau,\mathrm{Init}(p,\tau)) \in \mathcal{F} \cup C\} \supseteq \{a \mid \mathrm{In}(a,p',\tau',z') \in \mathcal{F} \cup C'\}$。

定义 5.25　对于 Σ_{dom}，规划 p 合理，当且仅当：

- 对于所有任务 $\tau \in \mathrm{Tasks}(p)$，$\mathrm{PlanTypes}(\mathrm{Init}(p, \tau))$ 中的所有规划类型都与 p 中的 τ 分层兼容。
- 对于所有状态 $z \in \mathrm{States}(p)$，$\cup \mathrm{PlanTypes}(z)$ 中的所有规划都合理。

因此，在递归式任务分配过程中，对于一个兼容规划，利用最多可用的智能体可推导出一个有效分配。该稳健性可保证在给定所参与智能体的信念基不发生冲突条件下，团队成员可就递归式任务分配的计算达成一致。

合理规划的概念并不能保证一定会得到完美分配，这需要利用以下两个定义所反映的更加充分的概念。

定义 5.26　规划类型 P 与规划 p 中任务 τ 是完全分层兼容的，当且仅当对于所有信念基$\mathcal{F}$，如果在假设条件$\mathcal{F}$下，对于规划 p 存在一个有效分配 C，且在假设条件$\mathcal{F} \cup C$ 下，对于规划 $p' \in P$ 也存在一个有效任务分配 C'，使得$\{a \mid \mathrm{In}(a,p,\tau,\mathrm{Init}(p,\tau)) \in \mathcal{F} \cup C\} = \{a \mid \mathrm{In}(a,p',\tau',z') \in \mathcal{F} \cup C'\}$。

定义 5.27　对于 Σ_{dom}，规划 p 完全合理，当且仅当：

- 对于所有任务 $\tau \in \text{Tasks}(p)$，$\text{PlanTypes}(\text{Init}(p,\tau))$ 中的所有规划类型都与 p 中的 τ 完全分层兼容。
- 对于所有状态 $z \in \text{States}(p)$，$\cup \text{PlanTypes}(z)$ 中的所有规划都完全合理。

这样，在一个完全兼容的规划中，所有可用智能体都将被分配，从而产生一个完美分配。

上述定义允许执行递归式任务分配的各种不同操作，从仅需要结果有效的基本情况，到完美分配的更高要求。给定所执行的规划合理或甚至完全合理，则可确保获得相应的结果，这将在 5.12 节讨论。

不论是合理还是完全合理，都可根据域和产生过程来创建。ALICA 程序的开发过程包括一个验证步骤，用于检查所建模规划的各自特性，而一个规划算法或学习算法限制只能产生一个合理规划或完全合理规划。通常，一个不合理的规划可通过其具体条件使之满足子规划条件而转化为一个合理规划。当然，这要求在父层次上所有参与的智能体都应具有相应的必要信息。

接下来，将讨论一种最优分配概念，使得所涉及的效用函数最大。在 5.12 节，将推导一个相应的任务分配算法。

5.10　最优式任务分配

在前面章节中，正式定义了任务分配问题及其有效解。在此，将重点考虑何为一个最优解以及该如何计算。

Nair 等人[104]对该问题进行了详细讨论，尽管将其看作角色分配，而并没有像本书一样将角色和任务区分开来。在其观点中，一个任务分配应考虑由事件或动作产生的潜在回报。一个全局最优任务分配还应考虑由于智能体损坏或情况发生变化而导致的进一步重分配。Nair 表明该问题是一个 NEXP 时间完备问题，因此难以处理。而一个局部最优任务分配仅考虑当前情况，仍是一个 NP 难题。Nair 同时还表明如何通过一个考虑潜在触发重分配的预运行计算来提高局部任务规划算法。然而在复杂领域中，由于规模随触发器个数和策略个数之积指数增长，导致该处理过程很快就不可行。另外，Nair 等人仅考虑独立回报，如假设将回报与执行某一特定任务的一个智能体相关联。而本书允许采用相互关联的效用函数，这依赖于相应规划的全分配。而且，在高动态领域，难以或甚至无法获得未来回报的必要的概率估计。最后为简化问题，本书仅考虑分层结构，其中每一层上不同的效用函数负责分配过程。这些效用函数不一定相关，但可相互推导。

为设计一个合适的通用且有效的分配算法，以下前提条件需成立：

● 回报问题留作以后解决。效用函数可考虑回报，但是针对特定域的。

● 可非常频繁进行分配，因此要求分配必须高效。

● 在运行期间，给定任务分配和相应情况的条件下，规划的效用函数必须完全可计算。这不依赖于子规划的分配。这种依赖关系可在预运行计算步骤中去除。

为此，本书将分层最优作为递归式任务分配问题解的衡量标准。

定义 5.28　在假设条件$\mathcal{F}$下，智能体 a 对状态 z 的一个递归式任务分配 C 是分层最优的，当且仅当 C 为最小集，使得对于 C 中所有规划类型 P，即$(\exists a', p, \tau, z') p \in P \wedge \mathrm{In}(a', p, \tau, z') \in C$：

$$\{\mathrm{In}(a', p', \tau, z') \mid \mathrm{In}(a', p', \tau, z') \in C \wedge p' \in P\} = \operatorname*{argmax}_{D} \max_{p \in P} \mathcal{U}(p)(\mathcal{F} \cup D)$$

满足 $\mathrm{TAlloc}_c^*(a, z \mid \mathcal{F})(C)$ 或 $\mathrm{TAlloc}_p^*(a, z \mid \mathcal{F})(C)$。

这样，一个分层最优任务分配将在保持完备性或完全完备性下（根据需要哪种特性），使得每个参与规划类型的效用函数最大。值得注意的是，如果只需有效性，即只要求 $\mathrm{TAlloc}^*(a, z \mid \mathcal{F})(C)$，则由于对智能体执行任何规划都没有激励，空分配为最优。这种激励可在没有完备性要求下选择规划来执行，然而选择何种规划执行应由规划自身结构（由规划算法提供）以及团队状态来确定。因此认为这样一种附加的选择机制是多余的，从而将完备性作为递归式任务分配所需的至少必要特性。

5.11　效用函数

除了前置条件和运行条件，效用函数反映了在某一特定情况下一个规划的适用性，并决定了任务分配。因此，其在 ALICA 中具有重要作用。如上节所述，一个任务分配的计算需要一个高效的搜索。在此对效用函数施加一种结构，以允许启发式估计。

首先，效用函数取决于特定域反映的当前情况和域独立函数反映的当前角色分配。

定义 5.29　函数 $\mathrm{pri}(p)$对规划 p 执行中所有智能体对各自当前任务的优先级进行评估：

$$\mathrm{pri}(p)(\mathcal{F}) = \begin{cases} -1 & \mathcal{F} \vdash (\exists a, \tau, r)\phi[a, p, \tau, t] \wedge \mathrm{Pref}(r, \tau) < 0 \\ \dfrac{1}{|\mathcal{A}|} \sum_{\mathcal{F} \vdash \phi[a,p,\tau,r]} \mathrm{Pref}(r, \tau) & \text{其他} \end{cases}$$

式中　$\phi[a, p, \tau, t] = \mathrm{HasRole}(a, r) \wedge (\exists z)\mathrm{In}(a, p, \tau, z)$。

也就是说，$\mathrm{pri}(p)$ 将智能体对其相应任务的所有优先级相加，然后利用因

子$\frac{1}{|\mathcal{A}|}$对结果归一化。若单个优先级小于零，则 pri(p) 定义为 -1，认为优先级为负表示无能力完成任务。

定义 5.30（效用函数）相对于信念基$\mathcal{F}$，一个规划 p 的效用函数$\mathcal{U}(p)$（$\mathcal{F}$）的形式为

$$\mathcal{U}(p)(\mathcal{F}) = \begin{cases} -1 & \text{pri}(\mathcal{F}) < 0 \\ w_0\text{pri}(\mathcal{F}) + \sum_{1\leqslant i\leqslant n} w_i f_i(\mathcal{F}) & \mathcal{F} | = \text{TeamIn}(p) \\ 0 & \text{其他} \end{cases}$$

效用函数是函数 pri，f_1，…，f_n在信念集上的加权和。权重 w_i为常数，使得 $\sum_{i=0}^{n} w_i = 1$ 且（$\forall i$）$0 \leqslant w_i \leqslant 1$。函数$f_i$：$2^{\mathcal{L}_S} \longmapsto [0 \cdots 1]$ 反映了特定域信息。根据 5.1 节中介绍的局部性原则，每个 f_i中的谓词 In(a, ρ, τ, z) 均发生时，ρ 可能仅表示 p。

根据该定义，效用函数仅限于加权和。然而由于 w_i可设为 0 且每个加项都可任意复杂，因此该结构并没有局限性。下节将表明阐述该结构并保持每个加项尽可能简单在实际中非常有用。

5.12 任务分配算法

一个任务分配算法计算了智能体的任务分配，表示为一个信念集。计算结果应满足定义 5.20 或定义 5.28 中分层分配的情况。

本书提出了一种基于由 Hart 等人[67]提出的 A* 搜索算法。与传统定义不同，在此采用最大效用函数而不是最小目标函数。为此，推导出效用函数$\mathcal{U}$的启发式估计$\mathcal{H}$。在每次迭代中，搜索过程通过将一个智能体分配给所有可能任务来扩展搜索节点。一旦所有智能体被分配并且根据所计算的分配满足规划条件，则完成一次分配。由于效用函数将智能体的信念状态映射到实数，因此启发式函数会将信念状态以及未分配的智能体都映射到实数：$H(p)$：$2^{\mathcal{L}_S} \times 2^A \longmapsto \mathbb{R}$。A* 算法的计算效率取决于启发式搜索的准确率和计算时间。

根据定义 5.30，给定一个效用函数，可用如下方式来推导启发式估计：在实现目标分配 G 的路径上，给定一个假设条件$\mathcal{F}$下的搜索节点，以及部分分配 H 和可用智能体 A，应满足：

$$\mathcal{U}(p)(\mathcal{F} \cup G) \leqslant \max_{H'} \mathcal{U}(p)(\mathcal{F} \cup H \cup H')$$

其中，H'完全分配 A 中所有智能体。另外，由于 $w_i \geqslant 0$，则

$$\max_{H'} \mathcal{U}(p)(\mathcal{F} \cup H \cup H') \leqslant w_0 \max_{H'} \text{pri}(\mathcal{F} \cup H \cup H') + \sum_{1\leqslant i\leqslant n} w_i \max_{H'} f_i(\mathcal{F} \cup H \cup H')$$

也就是说，通过逐个最大化每个加项，可构建一个启发式估计方法。再次提

醒与传统定义不同，该启发式方法包括效用函数，因此可估计一个局部节点的完全效用而不是潜在增益。这只是简化如下定义的一种技术手段。

在仅需要完全分配的情况下，$\max_{H'}\text{pri}(\mathcal{F}\cup H\cup H')$ 可由下式进一步估计：

$$h_{\text{pri}}(p)(\mathcal{F},A)=\begin{cases}-1 & \Phi[p,\mathcal{F},A]\\ \dfrac{1}{|A|}\Big(\sum_{\mathcal{F}\vdash\phi[a,p,\tau,r]}\text{Pref}(r,\tau)+\\ \sum_{a\in A}\max_{\tau:\psi[a,p,\tau,r]}\text{Pref}(r,\tau)\Big) & \text{其他}\end{cases}$$

式中 $\Phi[p,\mathcal{F},A]\overset{\text{def}}{=}\mathcal{F}\vdash(\exists a\in A,r)\text{HasRole}(a,r)\wedge(\exists z,\tau)\text{In}(a,p,\tau,z)\wedge\text{Pref}(r,\tau)<0$；

$\phi[a,p,\tau,r]\overset{\text{def}}{=}\text{HasRole}(a,r)\wedge(\exists z)\text{In}(a,p,\tau,z)$；

$\psi[a,p,\tau,r]\overset{\text{def}}{=}\mathcal{F}\vdash\text{HasRole}(a,r)\wedge\tau\in\text{Tasks}(p)\wedge(n,m)=\xi_t(p,\tau,\mathcal{F})\wedge m<|\{a'|\text{In}(a',p,\tau,z)\in\mathcal{F}\}|$。

也就是说，$h_{\text{pri}}(p)(\mathcal{F},A)$ 总结归纳了每个未分配智能体对 Tasks(p) 中任一任务的最大优先级。

如果要求分配结果应为完全完备的，则可采用更为严格的启发式方法，即

$$\Phi[p,\mathcal{F},A]\overset{\text{def}}{=}\mathcal{F}\vdash(\exists a,r)\text{HasRole}(a,r)\wedge(\exists z,\tau)\text{In}(a,p,\tau,z)\wedge\text{Pref}(r,\tau)<0\vee(\exists a\in A)\max_{\tau:\psi[a,p,\tau,r]}\text{Pref}(r,\tau)<0$$

以使启发式方法可反映未分配智能体必须分配给其角色具有正优先级的任务。

通常：

$$\max_{H'}\text{pri}(\mathcal{F}\cup H\cup H')\leqslant h_{\text{pri}}(p)(\mathcal{F},A)$$

由于 $h_{\text{pri}}(p)(\mathcal{F},A)$ 忽视规划条件，因此可能会对可能值估计过高。值得注意的是，用于完全分配问题的启发式方法也适用于完美完全分配问题，反之则不行。

特定域函数 f_i：$2^{\mathcal{L}_S}\longmapsto\mathbb{R}$ 由相应的启发式函数 h_i：$2^{\mathcal{L}_S}\times 2^A\longmapsto\mathbb{R}$ 进行估计。由于函数 f_i 可任意选择，因此难以对相应的启发式函数给出一个明确定义。

假设 T_i 为 f_i 中的任务集，则 h_i 可定义为

$$h_i(p)(\mathcal{F},A)=\max_{Q:\phi[Q,A,\mathcal{F},p]}f_i(p)(\mathcal{F}\cup Q)$$

式中 $\phi[Q,A,\mathcal{F},p]=Q\subseteq\{\text{In}(a,p,\tau,\text{Init}(p,\tau))|a\in A\wedge\tau\in T_i\}\wedge(\text{In}(a,p,\tau,z)\in Q\wedge\text{In}(a,p,\tau',z')\in Q\rightarrow\tau=\tau'\wedge z=z')\wedge(\forall\tau\in T_i)(\exists n,m)$

$\xi_t(p,t,\mathcal{F})=(n,m)\wedge|\{a|(\exists z)\text{In}(a,p,\tau,z)\in Q\cup\mathcal{F}\}|\leqslant m$。

理想情况下，每个函数 f_i 都取决于尽可能少的任务，从而可采用简单有效的启发式方法。尽管以上述方式构建的启发式方法能被接受，但大多数情况下，一种利用 f_i 自身结构所构建的启发式方法的执行性能明显更好。

现在，考虑下列示例：

示例5.2 示例5.1中厨师采用的表达式：

$$f(\text{ServeGuests})(\mathcal{F})=\frac{|\{a \mid \text{In}(a,\text{ServeGuests},\text{DeliverOrder},z)\in\mathcal{F}\}|}{100}$$

可自然地扩展为启发式形式：

$$h(\text{ServeGuests})(\mathcal{F},A)=\frac{|\{a \mid \text{In}(a,\text{ServeGuests},\text{DeliverOrder},z)\in\mathcal{F}\vee a\in A\}|}{100}$$

由此可产生一个比上述定义更加有效的启发式结构。

在机器人足球领域，一种常用方法是让最接近球的机器人执行进攻任务，这可用函数表示为

$$f(p)(\mathcal{F})=\max_{a:\text{In}(a,p,\text{Attack},z)\in\mathcal{F}}1-\frac{\text{Dist}(a,\text{ball})}{\text{maxDist}}$$

式中 Dist——两个物体之间的欧几里得距离；

maxDist——可能的最大距离，如足球场地的对角线。

启发式估计可通过对函数 f 进行简单扩展来定义：

$$h(p)(\mathcal{F},A)=\max_{a:\phi(a,p,\mathcal{F},A)}1-\frac{\text{Dist}(a,\text{ball})}{\text{maxDist}}$$

式中 $\phi[a,p,\mathcal{F},A]=\text{In}(a,p,\text{Attack},z)\in\mathcal{F}\vee(\xi_t(p,\text{Attack},\mathcal{F})=(n,m)\wedge$
$m<|\{a' \mid \text{In}(a',\text{Attack},p,z)\in\mathcal{F}\}|\wedge a\in A)$。

在需要传球的另一种情况下，传球机器人和接球机器人之间的距离非常关键，因此可定义一个函数为

$$f(p)(\mathcal{F})=\max_{a:\text{In}(a,p,\text{passing},z)\in\mathcal{F}}\ \max_{b:\text{In}(a,p,\text{Receiver},z)\in\mathcal{F}}1-\left(\frac{7\text{m}-\text{Dist}(a,b)}{\text{maxDist}}\right)^2$$

其中，设定理想的传球距离为7m。

相应的启发式函数会稍微复杂：

$$f(p)(\mathcal{F},A)=\max_{a:\phi[a,p,\mathcal{F},A]}\ \max_{b:\psi[a,b,p,\mathcal{F},A]}1-\left(\frac{7\text{m}-\text{Dist}(a,b)}{\text{maxDist}}\right)^2$$

式中：$\phi[a,p,\mathcal{F},A]=\text{In}(a,p,\text{Passing},z)\in\mathcal{F}\vee(\xi_t(p,\text{Passing},\mathcal{F})=(n,m)\wedge$
$m<|\{a' \mid \text{In}(a',\text{Passing},p,z)\in\mathcal{F}\}|\wedge a\in A$；
$\psi[a,b,p,\mathcal{F},A]=\text{In}(a,p,\text{Receiver},z)\in\mathcal{F}\vee(\xi_t(p,\text{Receiver},\mathcal{F})=(n,m)$
$\wedge m<|\{a' \mid \text{In}(a',\text{Receiver},p,z)\in\mathcal{F}\}|\wedge b\in A\wedge b\neq a)$。

上述示例表明尽管在某些情况下自动构建上述启发式函数可产生不错的结果，但在其他情况下，也可容易地手动构建一种更加有效的启发式函数。尤其是当每个加项中所涉及的任务个数较多时，高效的启发式函数非常关键。在此并不深入讨论启发式函数的自动构建，感兴趣的读者可参见参考文献［120，123］。

在给定所有单个启发式估计的条件下，可构造完备启发式函数：

$$H(p)(\mathcal{F},A)=\begin{cases}-1 & h_{\text{pri}}(\mathcal{F},A)<0\\ w_0h_{\text{pri}}(\mathcal{F},A)+\sum_{1\leqslant i\leqslant n}w_ih_i(\mathcal{F},A) & \phi[p,\mathcal{F},A]\\ 0 & \text{其他}\end{cases}$$

式中　$\phi[p,\mathcal{F},A]=\mathcal{A}|\geqslant\sum_{n:\tau\in\text{Tasks}(p)\wedge\xi_t(\tau,p,\mathcal{F})=(n,m)}\max(0,n-|\{a'|\text{In}(a',p,\tau,z)\in\mathcal{F}\}|)\wedge$

$|A|\leqslant\sum_{n:\tau\in\text{Tasks}(p)\wedge\xi_t(\tau,p,\mathcal{F})=(n,m)}m-|\{a'|\text{In}(a',p,\tau,z)\in\mathcal{F}\}|$。

根据效用函数和启发式函数，可定义一个基于 A^* 算法的搜索算法。一个规划类型的任务分配算法见表 5.2。假设对智能体、任务和规划进行总排序（需保证给定同样的信念状态），不同的智能体可计算同一分配，即使不同分配的效用函数相同。

该算法包括两个主要函数：InitTaskAlloc 和 NextAlloc。InitTaskAlloc 利用相应规划类型中每个规划的一个节点来初始化搜索顺序。每个节点包括一个局部分配和一个未分配智能体的有序列表。将尚未在一个规划中分配的智能体增加到每个节点的自由智能体列表中。

根据启发式方法，NextAlloc 对最优节点不断迭代扩展，直到找到一个满足 AllocGoal 的分配，或者直到节点序列为空，此时认为分配失败。随后调用 NextAlloc 函数，以效用降序返回假设条件$\mathcal{F}$下对规划类型的所有有效任务分配。

表 5.3 中的 Expand 函数反映了搜索节点之间的继承关系，并产生该搜索节点的子节点集。取决于整个问题是否可以找到一个完美完全分配，该函数可使得所有智能体均参与任务分配，或允许忽略智能体。

值得注意的是，同一规划类型中不同规划的分配是直接竞争的关系，即使效用函数可能不同。因此，从规划类型中如何选择一个规划的问题也是任务分配问题的一部分，这需要规划的效用函数具有可比性。对效用函数施加的结构和将有效分配对应于［0，1］，有助于设计具有可比性的效用函数。

表 5.2　一个规划类型的任务分配算法

```
InitTaskAlloc(P,F,Agents) {
  Queue_P := empty Queue;
  foreach(Plan p in P) {
    new Node n;
    n.Plan := p;
    foreach(Agent a in Agents) {
        if(F contains In(a,p,t,z) for some t,z) {
            Remove a from Agents;
        }
    }
    Add n to Queue_P;
  }
```

（续）

```
  foreach(Node n in Queue_P) {
    n.AgentsAvail := Agents;
    n.Alloc := ∅;
    n.Heuristic := H(n.Plan)(F, n.AgentsAvail);
  }
  Sort Queue_P by heuristic ;
}

NextAlloc(P,F) {
 while(Queue_P is not empty) {
    n := RemoveFirst(Queue_P);
    if (AllocGoal(n,P,F)) return n.Alloc;
    Add Expand(n) to Queue_P;
    Sort Queue_P by heuristic ;
 }
 return FAILURE;
}

AllocGoal(node,P,F) {
 if(node.AgentsAvail not empty or node.Heuristic < 0) return false;
 foreach(Task t in node.Plan) {
    (n_min,n_max) := ξ_t(node.Plan,t,F);
    n := |{a | In(a,node.Plan,t,z) ∈ node.Alloc ∪ F}|;
    if (n < n_min ∨ n > n_max) return false;
 }
 if (F ∪ node.Alloc ⊢ Pre(node.Plan) ∧ Run(node.Plan)) return true;
 else return false ;
}
```

表 5.3 任务分配的节点扩展函数

```
Expand(node,F) {
 if (node.AgentsAvail is empty) return ∅;
 a := RemoveFirst(node.AgentsAvail);
 nodes := ∅;
 foreach(Task t in node.Plan) {
    n := copy of node;
    Add In(a,node.Plan,t,Init(node.Plan,t)) to n.Alloc;
    (n_min,n_max) := ξ_t(node.Plan,t,F);
    n := |{a | In(a,node.Plan,t,z) in n.Alloc ∪ F}|;
    if (n ≤ n_max ∧ n+|n.AgentsAvail| ≥ n_min) {
        n.Heuristic := H(n.Plan)(F ∪ n.Alloc,n.AgentsAvail);
        Add n to nodes;
    }
 }
 if ( perfectly complete allocation not required ) {
    n := copy of node;
    n.Heuristic := H(n.Plan)(F ∪ n.Alloc,n.AgentsAvail);
    Add n to nodes;
 }
 return nodes;
}
```

命题1 如果对规划p任务分配算法返回一个分配结果，则该任务分配有效。

证明：该证明非常简单。首先，如果返回一个分配结果，则必然满足 AllocGoal 所定义的条件。因此其效用值大于0，从而由于满足基数，也必然满足规划p的 TeamIn(p)。另外，还满足规划p的前置条件和运行条件。由于对于某一智能体a和 Tasks(p) 中的某一任务τ，返回的分配结果中所包含的信念只有 In(a, p, τ, Init(p, τ))形式，并且只能对假设条件$\mathcal{F}$下未分配的智能体进行分配，因此分配结果也与$\mathcal{F}$和Σ_B相一致。

命题2 如果在假设条件$\mathcal{F}$下，利用智能体A对规划类型P存在一个有效任务分配，则分配结果必然会由 NextAlloc(P, $\mathcal{F}$) 返回。

证明：反证法，假设没有返回一个有效任务分配C。C将智能体A分配到一个规划$p \in P$，这样C既没有扩展也不满足 AllocGoal 中的条件。根据 InitTaskAlloc 的定义，创建一个规划p的初始节点，如果基数允许，随后的 Expand 函数将所有可用智能体分配给p中的任务，由于智能体集和任务集均有限，因此最终必须扩展C，并且所有有效任务分配均满足基数。

由于C有效，其效用值及其启发式估计都不小于0。另外还满足p中所有任务基数，以及假设条件$\mathcal{F}$和p的前置条件与运行条件。因此，也必然会满足 AllocGoal中的所有条件。

注意到定义5.19和表5.2均是指通过$\vdash$来证明前置条件和运行条件的一个智能体的能力。一个不完备的证明算法可能导致本应有效的分配无效。然而在实际中，这些条件通常非常简单，可由对应布尔值的几个必要函数表示。

给定一个任务分配算法，递归式扩展非常简单。表5.4给出了递归式任务分配算法。RecTaskAlloc 函数在给定状态z下将智能体分配给所有规划类型。如果采用该分配结果，应在分配智能体所处状态下递归调用其本身。

表5.4 递归式任务分配算法

```
Let a be the calculating agent
RecTaskAlloc(z,F) {
 A := {x | In(x,p,t,z) ∈ F};
 Q := ∅;
 foreach(PlanTypes P ∈ PlanTypes(z)) {
    InitTaskAlloc(P,F,A);
    do {
        G := NextAlloc(P,F);
        if (G = FAILURE) return FAILURE;
        zn := state of a in P according to G;
        if(zn exists ) H := RecTaskAlloc(zn,F∪G);
        else H:= ∅;
    } while(H = FAILURE);
    Q := Q∪G∪H;
 }
 return Q;
}
```

命题3　在假设条件$\mathcal{F}$下，智能体 a 在状态 z 利用 RecTaskAlloc 函数进行的任务分配是递归有效的。

证明：设 C 为递归式分配结果。由于根据命题1，通过调用 NextAlloc 函数可将 C 构建为任务分配联合，因此满足规划 a 的所有前置条件和运行条件，从而也满足定义5.21中的式（5.5）。根据命题1，式（5.6）和式（5.9）也同样满足。

由于 RecTaskAlloc 利用智能体 a 所分配给自身的状态 zn 递归简化规划结构，因此条件5.7满足。另外又由于 Expand 仅分配到初始状态，所以式（5.8）也满足。由于只对处于原始状态 z 的智能体进行分配，式（5.10）也满足。

最后，在同一规划类型中，智能体不能被分配两次，InitTaskAlloc 会排除所有已具有相应规划类型的智能体，并且对于某些智能体 a、规划 p、任务 τ 和初始状态 z，C 中仅包含形式 In$(a,\ p,\ \tau,\ z)$ 的字面意义，因此满足式（5.4）。

命题4　在假设条件$\mathcal{F}$下，智能体 a 在状态 z 利用 RecTaskAlloc 进行的任务分配是递归完备的。

证明：设 C 为递归式分配结果。根据命题3，C 为递归有效。由于 C 是由 NextAlloc 产生的有效分配的联合，并且 RecTaskAlloc 返回一个分配，当且仅当智能体 a 分配给所有状态，由于智能体 a 分配给每个规划中至多一个状态，因此 C 和$\mathcal{F}$可保证对每个规划类型智能体 a 在每个状态 a 下的分配都是一个有效分配。这满足定义5.22中的式（5.11）。

命题5　在假设条件$\mathcal{F}$下，智能体 a 在状态 z 利用 RecTaskAlloc 进行的任务分配是递归完全完备的。

证明：设 C 为递归式分配结果。首先，C 为递归完备的。由于如果需要的话，Expand 随后会对任务分配所有智能体，并且 ALLocGoal 要求未分配智能体的集合为空，因此任务分配问题的所有解可由 InitTaskAlloc$(P,\ \mathcal{F},\ A)$ 初始化，将 A 中所有智能体分配到 P 中的一个规划。由于 A 中只包含位于父状态的智能体，因此式（5.12）和式（5.13）满足。

命题6　如果存在一个递归完全完备的任务分配，则如果需要的话，RecTaskAlloc 会返回该任务分配。

证明：根据式（5.12）和式（5.13），一个递归完全完备的任务分配必须将智能体分配给所有涉及规划类型中的规划。由于 RecTaskAlloc 在每个规划类型（根据命题2）的所有有效任务分配中不断迭代，直到解决相应的递归问题，并且所有条件和效用函数都是局部规划，因此该命题成立。

命题7　如果存在一个递归完备的任务分配，则如果需要的话，RecTaskAlloc 会返回该任务分配。

证明：类似于命题6的证明，不同的是每一个单个分配步骤。根据 Expand，

NextAlloc 对所有可能的分配进行完全迭代以确定智能体。如果这些智能体中有一个满足 AllocGoal，则最终返回该分配。

由此，列表5.4中的算法会求得递归式任务分配问题的一个合理且完备的解。然而对于示例5.1中的餐厅场景，仍不能保证所有参与智能体达成一致。问题在于只满足 DeliverOrder 的规划类型在 ServeGuests 规划中与 DeliverOrder 任务并不完全兼容，从而 ServeGuests 规划不完全合理。但是，规划类型兼容，因此 ServeGuests 规划又是合理的。因此，在给定相同信念基的条件下，智能体 a、b 和 c 可对一个完备分配达成一致，但并不是一个完全完备的分配。

定义5.31　如果 $\cup\mathcal{C}$ 与 $\sum_B$ 一致，则分配集 $\mathcal{C}$ 也一致。

如果 $\cup_i\{\mathrm{In}(a,p,\tau,z) \mid \mathrm{In}(a,p,\tau,z)\in B_i \wedge p\in P\}$ 与 $\sum_B$ 一致，则具有信念基 B_i 的智能体集合 A 对规划类型 P 的所有分配均达成一致。

引理1　如果在相同假设条件 $\mathcal{F}$ 下，所有智能体 $a\in A$ 对规划类型 P 单独计算一个分配，则可得到相同的分配结果。

证明：根据每个智能体均采用相同假设，且 $\sum_B$ 为共同知识以及规划，任务和智能体以相同排序进行总排序，即可直接推导而得。

命题8　如果规划 p 是合理的，则对于智能体的所有有限集 A 以及规划 p 中的所有状态 z，A 与递归式完备分配 RecTaskAlloc(z, B) 一致。

证明：根据引理1，在给定相同的假设条件下，智能体对每个规划类型计算可得相同的分配。由于初始假设 B 相同，只需证明随着每个智能体递归计算分配，以同样方式来更新假设。由于所有规划仅包含局部规划的公式和效用函数，因此单个分配不依赖于其他单个分配。另外，由于规划 p 合理，只需在最高层（即对于 PlanTypes(z) 中的规划类型）调用可能失败的 NextAlloc。由于所有智能体都试图对 PlanTypes(z) 中的所有规划计算一个分配，根据引理1，这些智能体会达成一致。

命题9　如果规划 p 是完全合理的，则对于智能体的所有有限集 A 以及规划 p 中的所有状态 z，A 与递归式完全完备分配 RecTaskAlloc(z, B) 一致。

证明：类似于命题8的证明。

当然，这些条件只有在智能体具有相同信念基时相关，而这在现实情况下是很少发生的。信念冲突很容易导致冲突分配。在第6章，将介绍如何检测和消解冲突。

递归式任务分配作为一个重要工具，下节将介绍 pALICA 中基于规则的操作语义。接下来，用 TAlloc(a, P, $A\mid\mathcal{F}$) 来表示信念状态 $\mathcal{F}$ 下智能体集合 A 中智能体 a 对规划类型 P 的任务分配结果，以及用 RecTAlloc(a, $z\mid\mathcal{F}$) 表示信念状态 $\mathcal{F}$ 下智能体 a 对状态 z 的任务分配结果。另外，用 $\perp$ 表示寻找有效任务分配失败，例如，如果 RecTAlloc(a, $z\mid\mathcal{F}$) $=\perp$，则表明在给定假设条件 $\mathcal{F}$ 下对状态 z，

没有一个有效的任务分配。

值得注意的是，在计算分配时并没有限制是否需要完备或完全完备的特性。在 5.13 节将表明，这两种情况都可用于相同的操作规则，这取决于应用场景以及两种特性所用的规划结构。

5.13 规则

转移规则定义了在单个计算步骤中智能体配置如何变化。每条规则需满足一个条件，并将一个给定配置转化为新配置：

$$\text{规则名称：}\frac{\text{条件}}{\text{当前智能体配置}\rightarrow\text{新智能体配置}}$$

由于在一个给定情况下，可能会利用多条规则，在此引入优先级关系。该优先级决定了在一组应用规则中应采用哪条规则。如果规则 r_1 比规则 r_2 的优先级更高，则记为 $r_1 > r_2$。规则的优先级关系，即 $>$ 是可传递和非对称的。

下面，a 表示满足转移系统的智能体。仅当条件满足时才应用规则，即智能体配置 Conf(a) 与转移系统的左侧相结合且没有可应用的更高优先级的规则。

在此区分两种类型的转移规则：描述智能体正常操作的操作规则和提供从故障中恢复方法的修复规则。首先从 Init 规则开始执行一个 ALICA 程序，然后 Trans 和 STrans 规则描述智能体如何对规划中的转移做出反应。Trans 主要反映正常转移的情况，而 STrans 反映了同步转移的情况，这需要在所有参与机器人之间建立共同信念。之后是应用 Alloc 规则，该规则是对进入新状态的机器人进行递归式任务分配。最后，根据来自较低层（BSuccess）的成功信号以及在规划中达到成功状态（TSuccess），利用 BSuccess 和 TSuccess 规则来调整智能体的配置。在任何时刻，利用 Sense 规则来表示在信念基中集成了新的感知信息。

原则上，这 7 条规则已足以描述智能体的操作行为，除非发生故障，利用修复规则来进行故障处理。在此，提出了一组修复规则（10 条）来表示对故障的不同反应。由于某些规则并不能应用于任何领域，因此修复规则在某种程度上是域相关的。规则系统意味着仅适用于某个域的特定需要。

BAbort 规则是用于当出现故障时来终止正在执行的行为。BRedo 和 BProp 规则来处理故障。如果可能的话，BRedo 试图重新执行发生故障的行为，而 BProp 将该故障向上传递给包含行为执行发生故障的规划。PAbort 对规划的作用类似于 BAbort 对行为的作用，即终止发生故障的规划以及与其有关的所有规划和行为。然而如果可能的话，可由将失败规划中的智能体状态进行重置的 PRedo 来代替。这样可避免计算和通信的开销，因为智能体还继续执行任务，而无需重新计算一个新的分配。

在无法利用PRedo进行“软”重启且PAbort终止执行规划的情况下。PReplace规则触发一个新的任务分配，这个新的任务分配也可从相应的规划类型中重新选择一个规划。其他所有故障处理方法都不可行时，PProp将该故障向上传播到父规划。最后，PTopFail检测到顶层规划失败的情况，然后通过Init规则来重新初始化。

另外还有三种特殊规则。NExpand规则是当无法完成一个预期的任务分配时触发故障，Adapt规则是用于定期检查任务分配的效用值，并在当前效用值不再满足时执行任务的重分配，因此Adapt可用于高动态变化的分配，以适应环境的快速变化。最后，RoleAlloc规则用于处理角色分配和重分配。由于重分配通常发生在团队组成或某些功能发生变化的情况，因此也可将其看作一种修复规则。

5.13.1 操作规则

通常，操作规则的优先级要高于修复规则，这是基于通过改变智能体配置，使得故障可能无关紧要的思想。修复规则是为了保持某种特定状态或为当前执行但失败的意图提供另外一种选择。这也可允许通过指定一个转移条件下的失败子规划来进行特定域的故障处理。

初始化规则

$$\text{Init}:\frac{\top}{(B,\Phi,E,R)\rightarrow(B+\{\text{In}(a,p_0,\tau_0,z_0\},\text{Alloc}(z_0)\},\{(p_0,\tau_0,z_0)\},\Phi,R)}$$

从直观上看，初始化规则要求智能体开始执行规划树。这可能是由于启动后智能体的规划基为空或由于规划失败造成智能体的规划基完全为空。

感知规则

$$\text{Sense}:\frac{\text{Sense}(\phi)}{(B,\Upsilon,E,R)\rightarrow((B+\phi),\Upsilon,E,R)}$$

感知规则将感知信息集成到信念基。特定谓词Sense(ϕ)表示所获得的感知信息。假设通信信息也是以同样方式来集成，ALICA中智能体之间交换的一个关键消息是定期（半定期）广播规划基信息和成功信息。该消息中包含了对智能体a的规划基中每个三元组（p，τ，z）形式为In(a，p，τ，z）的信念和对不在该规划基中的每个三元组形式为¬In（a，p，τ，z）的负信念。这样就可通过对智能体a去除In(a，p，τ，z）中的所有信念来更新信念基，并增加在随后所接收的消息中所包含的信念。对于成功信息也可用相同方法处理。

大多数感知信息应由ALICA之外的感知部件来处理，并相应地更新智能体的信念状态，如5.2节所述。在此，出于完备性起见，仅列出该规则，使得可利用规则来描述所有相关的更新。

转移规则

转移规则控制智能体何时以及如何从一个状态转移到另一个状态。

$$\text{Trans}:\frac{B\vdash\phi\wedge(p,\tau,z)\in\Upsilon\wedge(z,z',\phi)\in\mathcal{W}\wedge\neg(\exists s\in\Lambda)(z,z',\phi)\in s}{(B,\Upsilon,E,R)\rightarrow((B-\vartheta_b^-)+\vartheta_b^+,(\Upsilon-\vartheta_p^-)\cup\vartheta_p^+,E',R)}$$

式中　$\vartheta_b^-=\{\text{In}(a',p,\tau,z)\mid B\vdash\text{In}(a',p,\tau,z)\}\cup\{\text{In}(a',p',\tau',z'')\mid a'\in\mathcal{A}\wedge p'\in\text{Plans}^+(z)\wedge z''\in\text{States}(p')\}$；

$\vartheta_b^+=\{\text{In}(a',p,\tau,z')\mid\text{In}(a',p,\tau,z)\in\vartheta_b^-\}\cup\{\text{Alloc}(z')\}$；

$\vartheta_p^+=\{(p,\tau,z')\}$；

$\vartheta_p^-=\{(p,\tau,z)\}\cup\text{Plans}^+(\Upsilon,z)$；

$E'=E-\{(b,z'')\mid(p',\tau',z'')\in\vartheta_p^-\}$。

如果智能体当前处于状态 z 且认为已满足转移条件 ϕ，则将按照一个外部转移从状态 z 转移到 z'。另外，该转移必须不属于同步集。发生转移之后应保证智能体终止执行与状态 z 相关（即 $\text{Plans}^+(\Upsilon,z)$ 中）的所有规划和行为。在信念基中增加 $\text{Alloc}(z')$ 可满足相对于新进入状态 z' 的任务分配需要。值得注意的是，应用该规则的智能体还假设处于状态 z 下的每个其他智能体也应用该规则，即认为满足其前置条件。通过一个规划树可实现对其他智能体的局部跟踪。

同步转移规则

同步转移规则用于处理同步集中的转移。直观上，一个同步转移对合作行为的启动进行建模，该合作行为取决于所有参与智能体在非常短的时间帧内动作。时间帧的上限取决于通信的延迟和可靠性，以及智能体跟踪其队友意图的精确度。在最坏的情况下，无法建立满足同步转移规则的条件。

$$\text{STrans}:\frac{(\exists A\subseteq\mathcal{A})a\in A\wedge(\exists s\in\Lambda)(z,z',\phi)\in s\wedge\psi}{(B,\Upsilon,E,R)\rightarrow((B-\vartheta_b^-)+\vartheta_b^+,(\Upsilon-\vartheta_p^-)\cup\vartheta_p^+,E',R)}$$

式中　$(p,\tau,z)\in\Upsilon$；

$\psi=(\forall(z'',z''',\phi_i)\in s)(\exists a'\in A,\tau'\in\text{Tasks}(p))B\vdash\text{MBel}_A(\text{In}(a',p,\tau',z'')\wedge\phi_i)$；

$\vartheta_b^-=\{\text{In}(a',p,\tau',z'')\mid a'\in A\wedge\tau'\in\mathcal{T}\wedge z''\in\text{States}(p)\}\cup\{\text{In}(a',p',\tau',z'')\mid a'\in A\wedge(B\vdash\text{In}(a',p,\tau',z''))\wedge p'\in\text{Plans}^+(z)\}$；

$\vartheta_b^+=\{\text{In}(a',p,\tau',z''')\mid a'\in A\wedge(B\vdash\text{In}(a',p,\tau',z''))\wedge(z'',z''',\phi)\in s\}\cup\{\text{Alloc}(z')\}$；

$\vartheta_p^+=\{(p,\tau,z')\}$；

$\vartheta_p^-=\{(p,\tau,z)\}\cup\text{Plans}^+(\Upsilon,z)$；

$E'=E-\{(b,z)\mid(p,\tau,z)\in\vartheta_p^-\}$。

在此，只有当智能体 a 确认其是智能体组 A 中的成员时才会进行同步转移，使得该智能体 a 认为在所有满足相关条件 ϕ_i 的集合 A 中具有一个共同信念。而且 a 还必须认为所有处于正确状态的集合 A 中的智能体具有一个共同信念，也就是说，同步集中的每个转移都将由一个智能体采用。因此，在进行同步转移的过

程中具有一个针对个体意图的共同信念。如果智能体认为该条件满足，则将以正常转移规则的方式进行动作。另外，还假设所有参与智能体都这么进行。

同步转移规则比正常转移规则的优先级高，即 STrans > Trans。这是根据同步实现规划对整个团队的作用更大且实现难度更大的思想。显而易见，如果不排除同步操作($\neg(\exists s \in \Lambda)(z,z',\phi) \in s$)的话，Trans 的条件可纳入 STrans 的条件。

顾名思义，同步操作实现了智能体之间的紧耦合。通过相互之间的通信可建立共同信念（详见 7.5 节）。在某些情况下，根据通信介质的质量和负荷，可能会适得其反。在这种情况下，可在普通转移的基础上采用弱同步的方式。假设要同步实现两个转移（z_1，z_2，ϕ）和（z_3，z_4，ψ）。假设 z_2，z_4 均不是初始状态，且没有产生 z_2 或 z_4 的其他转移，则可通过用 $\phi \wedge \psi \vee (\exists a, p, \tau)\mathrm{In}(a, p, \tau, z_4)$ 替换 ϕ 并用 $\phi \wedge \psi \vee (\exists a, p, \tau)\mathrm{In}(a, p, \tau, z_2)$ 替换 ψ 来实现一个类似于同步但又并非那么严格的行为，这样智能体就可依次实现同步转移。这种弱同步只需要一条消息，但更容易受包丢失或延迟的影响。

分配规则

分配规则用于智能体按上述转移规则发生转移之后的情况。通常当智能体刚刚进入一个状态时，该规则即可执行任务分配：

$$\mathrm{Alloc}:\frac{(B \vdash \mathrm{Alloc}(z) \wedge \mathrm{In}(a,p,\tau,z)) \wedge \mathrm{RecTAlloc}(a,z|B) \neq \perp}{(B,\Upsilon,E,R) \rightarrow ((B - \{\mathrm{Alloc}(z)\}) + \vartheta_b^+, \Upsilon \cup \vartheta_p^+, E \cup \vartheta_e^+, R)}$$

式中　$\vartheta_b^+ = \mathrm{RecTAlloc}(a,z|B)$；

$\vartheta_p^+ = \{(p',\tau',z') | \mathrm{In}(a,p',\tau',z') \in \vartheta_b^+\}$；

$\vartheta_e^+ = \{(b,z') | ((p,\tau,z') \in \vartheta_p^+ \vee z' = z) \wedge b \in \mathrm{Behaviour}(z')\}$。

也就是说，若智能体 a 认为对于状态 z 需要或可能执行一个任务分配，则将根据 5.12 节所述，通过计算分配 $\mathrm{RecTAlloc}(a, z|B)$ 来更新其信念基，然后将其涉及的所有规划都增加到规划基中，并开始执行所有与增加的规划—任务—状态三元组相关的行为。

由于直到智能体离开相应的状态，分配结果都是唯一相关的，因此转移规则要比分配规则具有更高的优先级，即 Trans > Alloc。

行为成功规则

在 ALICA 中，行为是不可分的动作，在执行过程中的任何时刻都可能失败或成功。这可由 $\mathrm{Success}(b)$ 和 $\mathrm{Fail}(b)$ 来反映，其中 b 为所讨论的行为。由于 b 内置在一个规范的行为规划或具有类似特性的规划中，通过转移规则，智能体将依次进入该规划的成功状态：

$$\mathrm{BSuccess}:\frac{(b,z) \in E \wedge B \vdash \mathrm{Success}(b)}{(B,\Upsilon,E,R) \rightarrow (B,\Upsilon,E - \{(b,z)\},R)}$$

式中　z——b 发生动作的状态。

如果成功执行一个动作行为，则该行为结束。根据规划公理，同时也满足其后置条件。由于该规则只是处理信号并相应地更新执行集，因此其优先级要高于之前修改规划基的 Trans 和 Alloc 规则。

任务成功规则

当且仅当智能体达到状态 $z \in \text{Success}(p)$，才能成功地完成规划中的一个任务：

$$\text{TSuccess}: \frac{(p,\tau,z) \in \Upsilon \wedge z \in \text{Success}(p)}{(B,\Upsilon,E,R) \rightarrow (B+\vartheta_b^+,\Upsilon,E,R)}$$

式中 $\vartheta_b^+ = \{\text{Succeeded}(a, p, \tau), \text{Post}(z)\}$。

通过将后置条件与达到终止状态相关联，该规则来更新信念基。同时还记录了成功完成 p 中的任务 τ。

如果由于基数 ξ_t（p，τ，$B+\vartheta_b^+$）改变而不再满足 TeamIn(p)，则该更新可能会导致一个不一致的分配。随后根据自适应规则的一个任务重分配可对智能体 a 进行相应的重分配，除非在父层上的一个转移作用于任务的成功完成。任务成功完成需要与执行规划 p 的其他智能体相互通信，否则一旦智能体 a 本身作用于成功，就可能由于执行规划的智能体个数不够而放弃执行（见示例 5.3）。这样，一个智能体就负责通知其团队由 Succeeded(a，p，τ）表示的类似于联合意图的实现目标。

在此，优先权优先赋于较低层次行为的成功实现，但仍认为该成功规则比转移规则具有更高的优先权，即 BSuccess > TSuccess > STrans。这样，对一个转移规则的检查可作用于任务的成功执行。

示例 5.3（异步规划成功） 该示例表明在执行某一规划的另一个任务之前，应先完成同一规划中的一个任务。如上所述，任务的成功完成会体现在信念基中，并与其他智能体相互通信。图 5.4 给出了这种情况。最初，两个智能体 a 和 b 分配给规划 p_1 中的任务 τ_1 和 τ_2。两个任务都有一个关联基数 1..1，表示在执行规划时只能给每个任务分配一个智能体。两个智能体都从该规划中各自任务的初始状态开始。智能体 a 执行规划类型 P_2，智能体 b 执行规划类型 P_3。经过一段时间，智能体 a 转移到下一状态 s_2。而智能体 b 在经过转移后从状态 s_3 到达成功状态 s_4。这时，智能体 b 执行完规划，因此也不再继续执行任务 τ_2。如果尚未记录成功实现任务 τ_2，则可能由于不满足任务基数而放弃规划。参见 TeamIn 的定义（定义 5.12）。通过发布成功消息，假设任务 τ_2 是 Required(p_1）中的唯一任务，则智能体 a 也由于 p_1 已成功实现而将放弃执行规划。

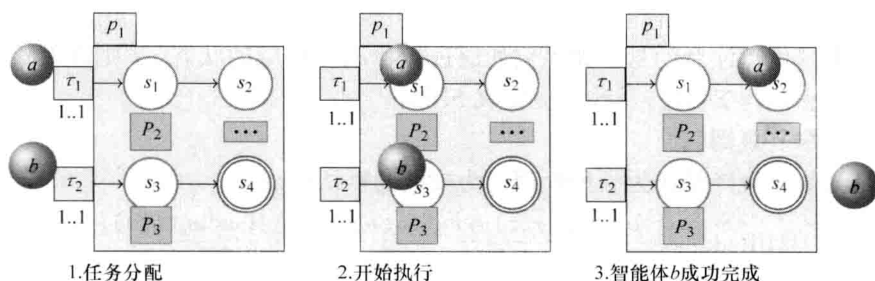


图 5.4　异步规划成功示例

5.13.2　修复规则

典型的 BDI 语言具有故障处理机制，如由于智能体工作的动态环境中发生不可预期的变化。经典的 BDI 语言还可区分规划失败和目标失败。前者可能是由于与某些边界约束冲突而导致，后者是由于目标本身不可能实现而导致。参见 Sardina 和 Padgham[141] 给出的示例。

尽管目标可表示为适当的后置条件，但在仅给定域相关知识的条件下，ALICA不具有推断目标是否仍可实现的任何推理能力。实际上，利用运行条件和失败状态等规划元素来表示未能实现目标或不可能实现目标的信念。

根据应用场景不同，经常会产生相应的规划失败概念。例如，在机器人足球动态领域，每个机器人应对队友的状态作出假设，由此来证明是错误的。修复规则是一种特殊的转移规则，意味着从失败或故障中恢复。处理失败的方法有多种，其中有一种根据域的特殊方法。可以重新尝试一个失败的规划，或由另一种规划替换，或可向规划树传播失败。在某些语言中，如 AgentSpeak[129]，甚至通过失败可产生某一特定目标，意味着为准确处理所发生的故障而触发一个自定义规划。

ALICA 具有一种显式处理规划失败的类似方法。首先确认失败的规划或行为，然后产生一个相应的信念并增加到信念基中（如5.2.2 节所述，对规划和行为分别为 $\mathrm{Handle}_f(p)$ 和 $\mathrm{Handle}_f(b,z)$）。由于转移规则的优先级要高于修复规则，因此可通过转移来建模一种显式机制，否则进行默认处理。

行为中止规则

如果产生一个故障信号，则行为中止，这样需要从执行集中去除该行为：

$$\mathrm{BAbort}:\frac{(b,z)\in E\wedge B\vdash \mathrm{Fail}(b)}{(B,\Upsilon,E,R)\rightarrow(B',\Upsilon,(E-\{(b,z)\}),R)}$$

式中　$B'=(B-\{(\forall i)\mathrm{Failed}(b,z,i),\mathrm{Fail}(b)\})+\{\mathrm{Failed}(b,z,j),\mathrm{Handle}_f(b,z)\}$；

$$j=\begin{cases}i+1 & \mathrm{Failed}(b,z,i)\in B\\ 1 & \text{其他。}\end{cases}$$

Failed(b, z, i) 信念用于记录需中止多少次行为，由此来采用不同的失败恢复规则。值得注意的是，如果例如通过转移离开相应的状态 z，则降低该信念以使得信念基与 Σ_b 保持一致（见定义 5.6）。

行为修复规则

行为修复规则是作为一种处理行为失败的默认机制：

$$\text{BRedo}:\frac{B\vdash \text{In}(a,p,\tau,z)\wedge \text{Failed}(b,z,1)\wedge \text{Handle}_f(b,z)}{(B,\Upsilon,E,R)\to(B',\Upsilon,E',R)}$$

式中 $E'=E\cup\{(b,z)\}$；

$B'=B-\{\text{Handle}_f(b,z)\}$。

$$\text{BProp}:\frac{B\vdash \text{In}(a,p,\tau,z)\wedge(\exists i)\text{Handle}_f(b,z)\wedge \text{Failed}(b,z,i)\wedge i>1}{(B,\Upsilon,E,R)\to(B',\Upsilon,E,R)}$$

式中 $B'=(B-\{\text{Handle}_f(b,z),\text{Failed}(b,z,i),(\forall k)\text{Failed}(p,k)\})+\{\text{Handle}_f(p),\text{Failed}(p,j)\}$；

$$j=\begin{cases}j'+1 & \text{Failed}(p,j')\in B\\ 1 & \text{其他。}\end{cases}$$

如果可能的话，BRedo 会重新执行一个失败的行为，并且如果开始尝试重新执行该行为，则 BProp 会向包含该行为的规划传播失败。在（$\forall k$）Failed(p, k) 中的通用量化可确保无论当前 k 值多大，都会去除 Failed(p, k)。该情况是每个规划 p 最多满足 Failed(p, k) 的一个实例。值得注意的是，这些规则的适用性是与具体的应用域有关。例如，在某些特定场景下，重新尝试一个失败行为可能没有任何意义。而在其他情况下，成功完成某一行为可与一个已知的概率分布相关联，这时就可对重试效用进行估计。因此，需要自定义这些域相关的规则。

规划中止规则

规划中止规则与行为中止规则非常类似：

$$\text{PAbort}:\frac{(p,\tau,z)\in\Upsilon\wedge(z\in\text{Fail}(p)\vee B\vdash\neg\text{Run}(p)\vee\neg\text{TeamIn}(p))}{(B,\Upsilon,E,R)\to((B-\vartheta_b^-)+\vartheta_b^+,\Upsilon-\vartheta_p^-,E-\vartheta_e^-,R)}$$

式中 $\vartheta_b^-=\{\text{In}(a',p,\tau',z')\mid a'\in\mathcal{A}\wedge\tau'\in\mathcal{T},z'\in\text{States}(p)\}\cup\{\text{In}(a',p',\tau',z')\mid(\exists z''\in\text{States}(p))p'\in\text{Plans}^+(z'')\}\cup\{(\forall k)\text{Failed}(p,k)\}$；

$\vartheta_b^+=\{\text{Handle}_f(p),\text{Failed}(p,j)\}$；

$\vartheta_p^-=\{(p,\tau,z)\}\cup\text{Plans}^+(\Upsilon,z)$；

$\vartheta_e^-=\{(b,z)\mid(p,\tau,z)\in\vartheta_p^-\}$；

$$j=\begin{cases}j'+1 & \text{Failed}(p,j')\in B\\ 1 & \text{其他。}\end{cases}$$

若与相应的运行条件相冲突，或到达一个失败状态，或智能体认为团队已不再执行规划，这时该规则就中止。另外，该规则会中止执行当前状态 z 下的所有规划

和行为，并假设所有在规划 p 中执行的智能体都可检测到规划失败，而且采用相同的规则。如果某个参与智能体没有检测到失败，就会产生信念冲突的状态，即某些智能体仍然认为该规划可继续执行，而其他智能体则认为无法执行。解决该问题需要智能体将相关失败信息通知给团队队友。定期广播包括足够信息的规划基可推测规划失败，或对缺失的智能体进行补偿。在 7.5 节中，将详细讨论这些消息。

规划修复规则

规划修复规则以默认方式处理失败规划。直观上，一个规划可重新执行，或由另一个规划替换，或向上传播失败：

$$\text{PTopFail}:\frac{B\vdash \text{Handle}_f(p_0)}{(B,\Upsilon,E,R)\rightarrow(B-\{(\forall p,\tau,z)\text{In}(a,p,\tau,z),\text{Handle}_f(p_0)\},\Phi,\Phi,R)}$$

PTopFail 通过重置智能体配置来处理顶层规划的失败，从而再次触发 Init 规则。解决该层次上失败的唯一方法是重新执行整个程序：

$$\text{PRedo}:\frac{(p,\tau,z)\in P\wedge z\in\text{Fail}(p)\wedge(B\vdash\neg(\exists x)\text{Failed}(p,x))\wedge B'\vdash\psi}{(B,\Upsilon,E,R)\rightarrow(B',(\Upsilon-\vartheta_p^-)\cup\{(p,\tau,z')\},E',R)}$$

式中　$B'=(B-\vartheta_b^-)+\{\text{In}(a,p,\tau,z'),\text{Alloc}(z'),\text{Failed}(p,1)\}$；

$\vartheta_b^-=\{\text{In}(a,p,\tau,z)\}\cup\{\text{In}(a,p',\tau',z'')\mid\tau'\in\mathcal{T}\wedge p'\in\text{Plans}^+(\Upsilon,z)\}$；

$z'=\text{Init}(p,\tau)$；

$\psi=\text{TeamIn}(p)\wedge\text{Pre}(p)\wedge\text{Run}(p)$；

$\vartheta_p^-=\{(p,\tau,z)\}\cup\text{Plans}^+(\Upsilon,z)$；

$E'=E-\{(b,z)\mid(p,\tau,z)\in\vartheta_p^-\}$。

通过采用 PRedo，如果某一智能体已处于失败状态，或认为其团队仍继续执行规划 p，或仍满足前置条件和运行条件，则该智能体可试图完成该规划 p 中的任务。值得注意的是，需要智能体处于采用 RPedo 之前的假设条件 B' 下来评估该规则的条件。PRedo 要比 PAbort 的优先级更高，即 PRedo > PAbort，因此首先应尝试计算量较小的失败处理。由于已将 Failed(p, 1) 增加到信念基中，因此 PRedo 不能处理同一规划的后续失败。处理顶层规划的失败只有一种方法，因此 PTopFail > PRedo：

$$\text{PReplace}:\frac{B\vdash\text{Handle}_f(p_0)\wedge\text{Failed}(p,1)}{(B,\Upsilon,E,R)\rightarrow(B',\Upsilon,E,R)}$$

式中　$B'=(B-\{\text{Handle}_f(p)\})+\{\text{Alloc}(z)\}$；

$p\in\text{Plans}(z)$。

PReplace 是通过在规划 p 执行的状态下触发一个新的任务分配来处理失败：

$$\text{PProp}:\frac{(B\vdash\text{Handle}_f(p)\wedge\text{Failed}(p,2))\wedge(p',\tau,z)\in\Upsilon}{(B,\Upsilon,E,\theta,R)\rightarrow((B-\vartheta_b^-)+\vartheta_b^+,\Upsilon-\vartheta_p^-,E-\vartheta_e^-,\theta,R)}$$

式中 $p \in \text{Plans}(z)$；

$\vartheta_b^- = \{\text{In}(a',p',\tau',z) \mid a' \in \mathcal{A} \wedge \tau' \in \mathcal{T}\} \cup \{\text{In}(a',p'',\tau',z') \mid (\exists z'' \in \text{States}(p'))p'' \in \text{Plans}^+(z'')\} \cup \{(\forall k)\text{Failed}(p',k)\}$；

$\vartheta_b^+ = \{\text{Handle}_f(p'),\text{Failed}(p',j)\}$；

$\vartheta_p^- = \{(p,\tau,z)\} \cup \text{Plans}^+(\Upsilon,z)$；

$\vartheta_e^+ = \{(b,z) \mid (p',\tau,z) \in \vartheta_p^-\}$；

$$j = \begin{cases} j'+1 & \text{Failed}(p',j') \in B \\ 1 & \text{其他。} \end{cases}$$

智能体对于规划失败的最后选择是将该失败向上传播，在此是通过中止父规划并触发相应的失败处理规则来实现的。由于失败应在最低层上进行处理，因此 PReplace > PProp。

分配失败规则

分配失败规则是用于处理一个任务分配无法对规划分配任何智能体的情况，例如不能满足前置条件。如果在状态 z 下的分配失败，则对相应的规划 p 也会失败：

$$\text{NExpand}: \frac{(B \vdash \text{Alloc}(z)) \wedge \text{RecTAlloc}(a,z \mid B) = \bot \wedge (p,\tau,z) \in \Upsilon}{(B,\Upsilon,E,R) \rightarrow ((B-\vartheta_b^-)+\vartheta_b^+,\Upsilon-\vartheta_p^-,E-\vartheta_e^-,R)}$$

式中 $\vartheta_b^- = \{\text{Alloc}(z)\} \cup \{\text{In}(a',p,\tau',z) \mid a' \in \mathcal{A} \wedge \tau' \in \mathcal{T}\} \cup \{\text{In}(a',p',\tau',z') \mid (\exists z'' \in \text{States}(p))p' \in \text{Plans}^+(z'')\} \cup \{(\forall k)\text{Failed}(p,k)\}$；

$\vartheta_b^+ = \{\text{Handle}_f(p),\text{Failed}(p,j)\}$；

$\vartheta_p^- = \{(p,\tau,z)\} \cup \text{Plans}^+(\Upsilon,z)$；

$\vartheta_e^- = \{(b,z) \mid (p,\tau,z) \in \vartheta_p^-\}$；

$$j = \begin{cases} j'+1 & \text{Failed}(p',j') \in B \\ 1 & \text{其他。} \end{cases}$$

自适应规则

自适应规则是针对规划没有失败但效用估计相对较低的情况。在这种情况下，如果智能体认为存在一个更适合于当前情况的分配，则可触发一个新的任务分配。这样就会产生一个类似于 Nair 等人[104]提出的局部决策的任务重分配机制。但不同的是，以该方式进行任务重分配可有效集成到编程语言，并可在无需过多审议的情况下随时进行。

若一个新分配的效用更高，那么就可用这个新的分配来替代原来的分配。为使得决策更加稳健，在此采用一个特定的规划阈值 $t(p)$，只有当两个分配的效用差大于该阈值时，才能进行任务重分配。另外，还采用了特定规划因子 $w_s(p)$ 加权的相似性度量 $\text{Sim}(p, B, B')$，使得不同的重分配必须具有更高的效用：

$$\text{Adapt}:\frac{\mathcal{U}(p_n)(B'')-w_s(p_c)\text{Sim}(p_c,B'',B)>\mathcal{U}(p_c)(B)+t(p_c)}{(B,\Upsilon,E,R)\to(B_n,\Upsilon_n,E_n,R)}$$

其中：

智能体 A 当前所处的状态 z：

$A=\{a'|\text{In}(a',p,\tau,z)\}$

智能体 a 在状态 z 下执行 p_c 或被动参与执行 p_c：

$a\in A\wedge(B\vdash\text{TeamIn}(A,p_c))\wedge p_c\in\text{Plans}(z)$

规划 p_c 和 p_n 属于同一规划类型：

$p_c\in P\wedge p_n\in P$

$\vec{P}_c$ 为重分配中子分支的规划：

$\vec{P}_c=\{p_c\}\cup\{p'|(\exists z'\in\text{States}(p_c))p'\in\text{Plans}^+(z')\}$

B' 是对 p_c 及其子规划没有任何假设的信念基：

$B'=(B-\{\text{In}(a',p',\tau',z')|a'\in A\wedge p'\in\vec{P}_c\}$

B'' 是结合假设 B' 对规划类型 P 的任务分配结果：

$B''=B'+\text{TAlloc}(a,P,A,|B')$

p_n 是有效任务分配选择的规划：

$\text{TAlloc}(a,P,A,|B')\neq\bot\wedge B''\vdash\text{TeamIn}(A,p_n)$

Δ_A^- 表示重分配的智能体：

$\Delta_A^-=\{a'|\text{In}(a',p_c,\tau',z')\in B\wedge\neg(\exists z'')\text{In}(a',p_c,\tau',z'')\in\text{TAlloc}(a,P,A|B')\}$

B_r 为保持不变的子分支的信念：

$B_r=\{\text{In}(a',p',\tau',z')|a'\in A\wedge a'\notin\Delta_A^-\wedge\text{In}(a',p',\tau',z'')\in B\wedge p'\in\vec{P}_c\}$

B''' 仅包含与 B_r 不冲突的信念：

$B'''=B''-\{\text{In}(a',p',\tau',z')|(\exists z'')\text{In}(a',p',\tau',z'')\in B_r\}$

B_n 为新的信念基，其中包含所有未变化的信念、新的任务分配以及智能体 a 认为对 z_n 至少重分配一个智能体，即经重分配后 a 所处于状态下的 $\text{Alloc}(z_n)$：

$$B_n=\begin{cases}B'''\cup B_r\cup\{\text{Alloc}(z_n)\} & \phi\\ B'''\cup B_r & \text{其他}\end{cases}$$

其中：

$\phi=\text{In}(a,p_n,\tau_n,z_n)\in(B'''+B_r)\wedge(\exists a')\text{In}(a',p_n,\tau_n,z_n)\in B'''$

针对新信念的规划基和执行集：

$\Upsilon'=\Upsilon-\{(p',\tau',z')|\text{In}(a,p',\tau',z')\notin B_n\}$

$E_n=E-\{(b,z')|(p',\tau',z')\in\Upsilon'\}$

如果智能体在该规则下进入一个新状态，则可在新规划基中体现：

$$\Upsilon_n = \Upsilon' \cup \{(p_n, \tau_n, z_n) \mid \mathrm{In}(a, p_n, \tau_n, z_n) \in B_n\}$$

$\mathrm{Sim}(p, \mathcal{F}, \mathcal{G})$ 为相似性度量：

$$\mathrm{Sim}(p, \mathcal{F}, \mathcal{G}) = 1 - \frac{|\{a \mid \mathrm{In}(a, p, \tau, z) \in \mathcal{F} \wedge (\exists z')\mathrm{In}(a, p, \tau, z') \in \mathcal{G}\}|}{\max(1, |\{a \mid (\exists p, \tau, z)\mathrm{In}(a, p, \tau, z) \in \mathcal{G}\}|)}$$

也就是说，如果在当前情况下，对规划类型 P 的一个新的任务分配的效用值要高于当前分配，则智能体采用新的分配。阈值 $t(p)$ 限制了该规则的适用范围。对于每个规划 p，$t(p)$ 和 $w_s(p)$ 都是具体的。值得注意的是，p_n 和 p_c 可以是同一规划类型 P 中的相同规划或两个不同规划。因此当某一任务分配不可行或难以实现时，该规则允许智能体从一组分配方案切换到另一种方案。与转移规则类似，需从信念基、规划基和执行集中去除当前分配。同时与任务分配规则相类似，可直接采用新分配。由于当与运行条件发生冲突时，该规则还实现了一种软修复机制，因此要比所有修复规则的优先级更高。

信念更新的定义反映了变化最小的概念，即无需改变任务的智能体也不会改变其状态。一旦采用 Adapt 对局部智能体进行重分配，或认为已重分配其他智能体到该状态，则可将信念 $\mathrm{Alloc}(z_n)$ 添加到信念基，并触发随后的递归式任务分配。

Adaptation 规则可使团队对发生变化的情况作出快速反应。根据阈值和相似性度量，可避免产生振荡。然而这些值取决于所采用的效用函数，因此该规则是域相关的。另外，这些阈值和相似度权重还可对分别每个规划允许或禁止用于动态自适应。通常，随着智能体逐步趋向于成功完成任务，整个效用值也会不断增大。在给定成功完成的回报函数条件下，实现上述目的的一种方法是采用满足 Bellman 方程[8]的效用值。

角色分配规则

角色分配规则是用于处理需要进行角色重分配的情况。根据队形 F（见定义 5.15）计算重分配，并在与之前分配情况不同时由智能体采用：

$$\mathrm{RoleAlloc}: \frac{|\{\mathrm{HasRole}(a', r) \mid \mathrm{HasRole}(a', r) \in B\} \neq F(\{a' \mid \mathrm{In}(a', p_0, \tau_0, z_0) \in B\})}{(B, \Upsilon, E, R) \rightarrow ((B' - \vartheta_b^-) + \vartheta_b^+, \Upsilon, E, R')}$$

式中　$R' = \{\mathrm{HasRole}(a, r) \mid \mathrm{HasRole}(a, r) \in F(\{a' \mid \mathrm{In}(a', p_0, \tau_0, z_0) \in B\})\}$；

$\vartheta_b^- = \{(\forall a, r)\mathrm{HasRole}(a, r)\}$；

$\vartheta_b^+ = F(\{a' \mid \mathrm{In}(a', p_0, \tau_0, z_0) \in B\})$。

如 5.6 节所述，只有在由于智能体脱离或加入团队，或由于某个智能体能力发生变化而造成团队组成发生变化的情况下才进行角色的重分配。为简单起见，在规则中没有明确表示出相应的触发条件。

整个团队是由顶层规划中处于孤立状态的智能体集合在一个智能体的信念中进行表示，即 $\{a' \mid \mathrm{In}(a', p_0, \tau_0, z_0)\}$。由此，当该智能体集合发生变化或智能体

得知任何一个参与智能体的能力发生变化时，这足以验证该规则的适用性。

总之，在此所提出的修复规则可构成一个灵活且可定制的系统来对失败作出反应，从而使得对于变化的情况，团队行为具有鲁棒自适应性。表 5.5 中按照规则的优先级顺序，对所介绍的规则进行了简要概括。

表 5.5　从最高优先级到最低优先级的 pALICA 操作规则

规则	说明
Init	初始化或重新初始化智能体配置
RoleAlloc	根据一个新计算的角色分配来更新智能体的假设角色
Sense	将新的感知信息集成到信念基
BSuccess	对行为成功完成作出反应
TSuccess	对任务成功完成作出反应
STrans	建立一个合适的共同信念之后，使得智能体沿同步转移运动
Trans	智能体从一个状态运动到另一个状态
Alloc	在智能体进入的状态中实现递归式分配
Adapt	若相应任务分配的效用值之差较大，则用新的任务分配代替当前的任务分配
BAbort	中止一个失败行为的执行，并设置相应标志
BRedo	如果可能，重新执行失败行为
PTopFail	处理顶层规划 p_0 失败的情况
PRedo	如果可能，重新执行一个失败规划中的任务
PAbort	中止一个失败规划的执行
PReplace	通过触发父状态中的一个新的任务分配来代替失败规划
PProp	将失败规划传播到其父规划
NExpand	无法计算一个失败规划的有效任务分配

从满足执行一个 ALICA 程序所需的最小规则集的意义上，在此所提出的规则集并非一个最小集。例如，PRedo 和 BRedo 规则并不是必需的，或 Alloc 和 Adapt规则可合并为一个规则。然而该规则集比最小规则集更易于修改，例如在无需动态自适应或甚至适得相反的域中，可很容易地去除 Adapt 规则。

5.14　智能体配置一致性

本节将验证定义 5.4 中所介绍的规划基公理的操作语义，另外还表明智能体的规划基总是构成一个包括所有行为的树型结构，即与所执行行为相关的均在规划基中，并且智能体总是信任其所进行的工作。根据定义 5.8，最后一个特性是信念基所需的。

根据定义 5.8，一个信念基需要与共同知识和规划基保持一致。

命题 10　如果 $\text{Conf}(a)=(B,\Upsilon,E,R)$，则当且仅当 $B\vdash \text{In}(a,p,\tau,z)$ 时，$(p,\tau,z)\in\Upsilon$。

证明：初始的信念基中并不包含 $\text{In}(a,p,\tau,z)$ 形式的任何信念，且根据定义 5.2，初始规划基为空。调整规划基的每个规则也会等效地更新信念基。

由此，信念基与规划基保持一致，且智能体总是信任其所要进行的工作。如 5.2.3 节所述，可通过信念更新语义来保持共同知识的一致性。

命题 11　如果程序结构良好，在运行期间产生的所有规划基中，每个规划 p 至多包含一个三元组（p，τ，z）。

证明：下面，假设一个满足每个规划中至多包含一个三元组的规划基。初始的规划基为空，因此满足该特性，然后通过应用一些规则对规划基进行修改。利用 Init 规则对初始的空规划基增加一个三元组，因此产生一个符合要求的规划基。

Sense、TSuccess、BAbort、BRedo、BProp、BSuccess、PReplace 和 RoleAlloc 规则并不会修改规划基。PAbort、PTopFail、PProp 和 NExpand 规则只是从规划基中去除三元组，同样也满足要求。PRedo 规则对规划基增加了一个三元组（p，τ，z'），但同时会去除（p，τ，z），从而也满足要求。

Trans 和 STrans 规则会去除转移发生时规划 p 和状态 z 中的三元组，以及属于分支 $\text{Plans}^{+}(z)$ 的所有三元组，而只增加一个规划 p 的新的三元组。这样如果输入规划基满足要求，则这两个规则都会产生符合要求的规划基。

Alloc 规则用于递归式任务分配，并将所得的三元组增加到规划基。由于程序为树形结构，递归式任务分配算法对每个规划一次至多分配一个智能体（根据命题 3），并考虑当前信念，该信念满足命题 10 中的输入规划基，则如果相应的输入规划基满足要求，Alloc 规则所产生的所有规划基也都会满足要求。

在对局部智能体进行重分配时，Adapt 规则会去除子分支相关的所有信念，随后还会去除规划基中的相应三元组。由于 $\text{TAlloc}(a,A,P\mid\mathcal{F})$ 只会产生规划 P 的有效分配，因此所增加的唯一三元组是针对智能体进入的新状态。如果增加了该三元组，则与之前执行规划有关的任何三元组都会被去除。因此，该命题对于 Adapt 规则同样满足。

命题 11 是规划基在任何时候构成一个树型结构的必要条件，但不是充分条件。

命题 12　对于规划基 Υ 中的任何元组（p，τ，z）；

- $p=p_0$；
- 规划基 Υ 中的其他元组（p'，τ'，z'）使得 $p\in\text{Plans}$（z'）。

证明：下面，假设一个符合要求的规划基。初始的规划基为空，因此满足要求。Init 规则会对初始的空规划基增加一个元组（p_0，τ_0，z_0），同样也满足该命题。Sense、TSuccess、BAbort、BRedo、BProp、BSuccess、PReplace 和 RoleAlloc

规则均不会修改规划基。Trans 和 STrans 规则会去除智能体离开状态时所执行的所有三元组，从而也满足该命题。

由于 RecTAlloc 仅产生有效的递归式任务分配，因此 Alloc 规则会对规划基增加一个完整的分支。Adapt 规划在规划类型 P 中重新分配。在规划基中去除完整分支，并增加一个规划 p_n的三元组。由于 p_n与所替换的规划 p_c都属于同一规划类型，并且三元组（p，τ，z）保持不变，因此所得的规划基仍满足要求。

PAbort、PProp 和 NExpand 规则会去除中止规划中所执行的所有规划。PTop-Fail 规则会产生一个空的规划基。PRedo 规则会去除（p，τ，z）以及状态 z 相关的所有元组，并增加一个三元组（p，τ，Init(p，τ)）。这样如果输入规划基满足命题要求，则所得的规划基也同样满足。

定理 5.1 如果 ALICA 程序结构良好，则运行过程中的所有规划基会构成一个树型结构。

证明：根据命题 11 和命题 12，以及一个结构良好的程序要求规划树为一个树形结构，可证明该定理成立。

要求规划基构成一个树型结构的特性非常重要，由此可简化推理，允许通过合适的数据结构来实现，并有助于高效遍历和规则应用。

在给定规划基为树型结构的条件下，可保证该规划基满足规划基公理（见定义 5.4）：

定理 5.2 如果 ALICA 程序的结构良好，则运行过程中的所有规划基均满足规划基公理 Σ_p。

证明：根据命题 11，可表明满足公理 5.1。另外，由于下列原因，可满足公理 5.2：

● p_0不属于程序树中的一个规划类型，否则程序不能构成一个树型结构，因此 Init 规则满足。

● 由于程序构成了一个树型结构，RecTAlloc 仅针对每个规划类型中的一个规则进行分配。

● 如果 Adapt 规则增加一个三元组，则会去除之前所执行规划的三元组。

● 没有其他规则可以开始执行一个新规划，即增加一个元组（p，τ，z），从而在输入规划基中没有三元组（p，τ'，z'）。

由于下列原因，可满足公理 5.3：

● 空规划基满足公理。

● 三元组（p_0，τ_0，z_0）满足公理的条件。

● 仅当对于 ϕ 存在转移（z，z'，ϕ）时，Trans 和 STrans 规则才会去除三元组（p，τ，z），并增加元组（p，τ，z'），因此这两个规则都满足要求。

● 任务分配算法仅针对相应规划的任务进行分配，即 Tasks(p) 中的一个元

素，且仅在初始状态下进行分配，因此 Adapt 和 Alloc 规则满足要求。

- 没有其他规则会改变智能体的任务或对于某个规划 p 和某个任务 τ 增加一个（p，τ，$\mathrm{Init}(p,\tau)$）之外的三元组，从而使得（p，τ，z）处于某个状态 z 下的输入规划基中。

除了满足规划基，还要保证机器人只会执行其想要进行的动作。也就是说，机器人只能执行由规划基表示的反映过程意图的行为。

命题 13 没有一个行为是孤立的。对于运行过程中的任一智能体配置（B，Υ，E，R），满足：

$$(\forall(b,z)\in E)(\exists p,\tau)(p,\tau,z)\in\Upsilon$$

证明：通过 Alloc 和 BRedo 规则可将行为增加到执行集。BRedo 规则要求通过 $\mathrm{In}(a,p,\tau,z)$ 使得智能体处于相应状态，而 Alloc 规则只增加元组（b，z），从而使得一个相应的三元组（p，τ，z）会同时增加到规划基中。

通过 Trans、STrans、PAbort、PTopFail、PRedo、PProp、NExpand 和 Adapt 规则，会在规划基中去除三元组。除了会产生一个空执行集的 PTopFail 规则外，上述这些规则都会从针对规划基中所去除元组的状态下的执行集中去除所有元组。

随后，如果一个行为是在状态 z 下进行，则智能体也认为其处于状态 z。

5.15 本章小结

本章介绍了 pALICA 语言的形式化语义。在此讨论了一个语义表示的智能体模型，这是通过一个由信念基、规划基、执行集和角色集构成的智能体配置。信念基反映了智能体所具有关于环境及其团队的当前信息。对于所执行的 ALICA 程序，智能体的内部状态是由构成智能体意图的过程描述的规划基来表示的。执行集是由智能体当前执行的行为组成的，从而与底层执行部件相关联。最后，角色集包含了团队中一个智能体可能具有的所有角色。

通过一个详细描述了智能体配置如何以及何时更新的转移规则系统来阐述语义的主要部分。规则集包括指导智能体正常操作的操作规则和针对失败和环境发生不可预见的变化作出反应的修复规则。对于任务分配问题的求解，介绍了一种递归式分配算法，即在规划层次的每一层中对智能体分配任务。根据局部性原则和自治性原则，针对所参与的规划，每个智能体对分配进行局部计算。该方法要求对于给定的 ALICA 程序，必须对智能体完成计算分配的一个合理特性。根据规划之间的耦合程度，区分并讨论了两种不同需求。最后，表明如果根据第 4 章所述程序是结构良好的，则每个智能体的内部状态都会构成一个树，并满足必要的一致性公理。为提高效率，可通过具体实现来利用该特性。

第 6 章　冲突检测与消解

在 ALICA 中，团队合作的冲突主要体现在任务分配的冲突上。对于参与的每个规划，每个智能体都具有一组表示对该规划中的任务分配智能体的信念。正如 5.8 节所述，分配冲突可能是由于团队中的信任基不同、执行规划不合理或执行不允许闲置的不完全规划（即强制执行完全分配）而造成的。

在参考文献［158］中，讨论了一种不依赖域相关信息来检测和消解这种冲突的方法，本章主要是基于上述研究成果。回顾定义 5.18 和定义 5.19，将任务分配看作一个 $\mathrm{In}(a, p, \tau, z)$ 形式的信念集。通过一种任务分配算法，对规划类型或层次化规划类型进行任务分配计算，之后根据所接收的信息来更新相应的信念。由于 ALICA 智能体定期广播其规划基，因此根据远端计算的部分结果来更新本地计算的任务分配。另一种方法是直接根据行为识别来进行更新，这样如果本地智能体可观测到其他智能体的动作，则可推理其在规划树中的状态[75]。在此，将这种识别任务看作一种环境通信。

6.1　冲突检测

由于是对于一个规划或规划类型产生冲突，因此接下来仅针对一个特定的规划类型 P 进行分配，这是限于 $\mathrm{In}(a, p, \tau, z)$ 形式的信念基子集，其中 $p \in P$。另外，由于在随后的冲突检测机制中状态并不重要，在此引入宏 $\mathrm{In}(a,p,\tau) \overset{\mathrm{def}}{=} (\exists z)\mathrm{In}(a,p,\tau,z)$。

定义 6.1（冲突）　两个分配 C_1 和 C_2 发生冲突，当且仅当两者都是针对同一规划类型 P，并且对于某一智能体 a，不满足 $C_1 \vdash \mathrm{In}(a, p, \tau) \leftrightarrow C_2 \vdash \mathrm{In}(a, p, \tau)$。对于规划类型 P，两个智能体 a_1 和 a_2 发生冲突，当且仅当两个智能体均信任冲突分配。在此，将 $\mathrm{In}(a, p, \tau)$ 称为冲突的一个成因。值得注意的是，产生冲突可能会有多种成因。

命题 14　如果对于规划类型 P，由 $\mathrm{In}(a, p, \tau)$ 造成两个智能体 a_1 和 a_2 发生冲突，那么 a_1 或 a_2 和智能体 b 之间必有冲突。

证明：设对于规划类型 P，两个智能体 a_1 和 a_2 发生冲突。不失一般性，进一步假设智能体 a_1 信任 P 的一个任务分配，其中包含冲突成因的某个状态 z 下的 $\mathrm{In}(a, p, \tau, z)$。则对于智能体 a_2 信任的 P 中的任务分配 C_2，必然满足：

- b 根本没有分配，即$(\forall p',\tau',z')\mathrm{In}(b,p',\tau',z')\notin C_2$；
- b 分配给 p 中另一个任务 τ'，即$(\exists \tau',z')\mathrm{In}(b,\tau',p,z')\in C_2 \wedge \tau\neq\tau'$；
- b 分配给 p 中另一个规划 P，即$(\exists p',\tau',z')\mathrm{In}(b,p',\tau',z')\wedge p\neq p'$。

根据规划基的一致性（定理 5.2），b 不能分配给同一规划中的两个任务或同一规划类型中的两个规划，因此无法保持不与 a_2 冲突的信念 $\mathrm{In}(b, p, \tau, z)$。由于不能保持与 a_1 冲突的 $\mathrm{In}(b, p, \tau, z)$，因而该命题成立。如果 b 为 a_1 或 a_2，该命题也同样成立。

该特性可保证在团队中分散分配的智能体至少会与一个智能体产生冲突，由此具有冲突检测的机会。

为设计一种检测机制，需要观察任务分配是如何随时间变化的。对于规划类型 p，一个智能体的分配会根据下列情况发生变化：

- 重新分配：智能体采用一个新的分配来提高效用。
- 消息：智能体接收一个有关 p 中其他智能体状态的消息。
- 放弃：智能体中止规划，由此不再执行分配。
- 删除：某些时候，智能体没有接收到来自其他智能体的消息。这种情况下，从团队中去除该智能体，即不再有任务分配。

删除是考虑到智能体损坏并失去通信能力而采取的。由于删除事件可看作消息事件的一种特殊情况（即一个空规划基消息），在此将不作考虑。同样，也不再考虑智能体放弃规划的事件，因为该事件会终止本地智能体执行规划，通过消息事件这对于其他智能体是可见的。通过智能体不断重复参与或放弃一个规划，可检测由此产生的冲突。持续冲突（即持续较长时间的冲突）会导致在智能体所信任的分配中产生一个时空格局时间模式，因此在此正式定义分配是如何随时间变化的。

定义 6.2（分配事件） 一个分配事件 e 是一个由分配加ϑ^+和分配减ϑ^-组成的二元组（ϑ^+，ϑ^-），两者都是形式 $\mathrm{In}(a, p, \tau)$ 的基础集合，使得$\vartheta^+\cap\vartheta^-=\Phi$ 以及$(\exists P)(\forall p)\mathrm{In}(a,p,\tau)\in\vartheta^+\cup\vartheta^-\rightarrow p\in P$，即所有分配事件都仅包含对单个规划类型 P 的加和减。

利用二元仿 $\circ$ 来表示分配事件的组合，即 $e_1\circ e_2=e_3$。

定义 6.3（分配事件组合） 设 $e_1=(\vartheta_1^+,\vartheta_1^-)$ 和 $e_2=(\vartheta_2^+,\vartheta_2^-)$ 为两个任务分配事件，则其组合可定义为

$$e_1\circ e_2\overset{\text{def}}{=}((\vartheta_1^+-\vartheta_2^-)\cup(\vartheta_2^+-\vartheta_1^-),(\vartheta_1^--\vartheta_2^+)\cup(\vartheta_2^--\vartheta_1^+))$$

该组合允许对所采用分配中多个事件的结果进行独立推理。在事件组合下，分配事件构成一个可交换群。

如果一个智能体 a 从另一个智能体 b 接收到一个规划消息，这会产生一个针

对智能体 a 正在执行的每个规划类型的分配事件。其中有四种不同情况：

- 对于分配，智能体 a 与 b 达成一致，因此分配事件为空，可忽略。
- a 不信任 b 所参与的规划类型，而 b 信任，这对于某些 p 和 τ 将产生分配事件($\{In(b, p, \tau)\}$, Φ)。
- a 认为 b 所执行的任务或规划与 b 实际执行的规划或任务不同，由此产生事件($\{In(b, p, \tau)\}$, $\{In(b, p', \tau')\}$)，使得 $p \neq p' \vee \tau \neq \tau'$。
- a 误认为 b 参与了该规划类型，则对于某些 p 和 τ 会产生分配事件(Φ, $\{In(b, p, \tau)\}$)。

根据命题 14，这些是所需考虑的情况。如果对 a 进行重分配，且两个事件的组合中不含有关于 b 的情况，则会产生一个循环。

定义 6.4（分配循环）　设 e_m 为一个由于从智能体 b 发送给智能体 a 的规划消息的非空分配事件，并设 e_r 为 a 接收该消息后的下一个分配事件。如果 $e_m \circ e_r = (A,B)$，使得 $(\forall p,\tau) In(b,p,\tau) \notin (A \cup B)$，则会产生一个循环。

因此，如果智能体根据重新分配的消息可恢复分配，则发生一次循环。智能体可很容易地通过监视分配事件来检测循环。当然循环并不意味着会有冲突，也可能循环只是反映了一个快速变化情况。然而如果连续发生多次循环，则将不像是对情况变化的正常反应。循环周期长度取决于智能体通信和思考的频率。在机器人足球比赛中，通常平均通信频率为 10Hz，而思考周期频率为 30Hz，这样两次连续循环的持续时间平均为 117ms。在这段时间内，要求团队来回改变两次分配的情况是不大可能的。因此可限制连续循环的次数小于 2，否则若超过两次，则智能体有信心来假设产生冲突。只要相对于动态环境，通信频率和思考频率较高，且接收到发送消息的概率不为 0，就可设置一个合理的限制。

接下来，将循环周期检测集成到 pALICA 的操作语义中。记 $\Delta(P, B_1, B_2)$ 来表示规划类型 P 中两个信任基 B_1 和 B_2 之间的分配不同：

$$\Delta(P,B_1,B_2) \overset{\text{def}}{=} (\{In(a,p,\tau) \mid p \in P \wedge (\exists z) In(a,p,\tau,z) \notin B_1 \wedge In(a,p,\tau,z) \in B_2\},$$
$$\{In(a,p,\tau) \mid p \in P \wedge (\exists z) In(a,p,\tau,z) \notin B_2 \wedge In(a,p,\tau,z) \in B_1\})$$

为检测循环周期，智能体必须记录部分分配历史。假设在信任基中具有两个谓词：Cycles(P, i) 表示规划类型 P 执行过程中发生的第 i 次循环，ADiff(P, d) 表示在规划类型 P 中记录的当前分配不同 d。

在操作规则的形式中进行必要更新，以扩展 5.13 节中定义的原始操作语义。写出：

$$t: \frac{Conf(a) \xrightarrow{v} Conf(a)'}{(B,Y,E,R) \rightarrow (B',Y',E',R')}$$

表示如果规则 r 将配置 Conf(a) 映射到 Conf(a)′，则规则 t 将(B, Y, E, R)

映射到（B'，Υ'，E'，R'）。

首先，只要一个智能体通过接收消息或行为识别获悉有关其他智能体状态的新信息，规则 Sense 就将该信息集成到信任基中。规则 Sense_{cd} 则是对此进行扩展，来更新由 ADiff 表示的历史信息：

$$\text{Sense}_{cd}:\frac{(B,\Upsilon,E,R)\xrightarrow{\text{Sense}}(B',\Upsilon',E',R')}{(B,\Upsilon,E,R)\rightarrow((B'-\vartheta^{-})+\vartheta^{+},\Upsilon',E',R')}$$

式中 $\vartheta^{-}=\{\text{ADiff}(P,d)\mid(p,\tau,z)\in\Upsilon\wedge p\in P\}$；

$\vartheta^{+}=\{\text{ADiff}(P,d\circ\Delta(P,B,B'))\mid\text{ADiff}(P,d)\in\vartheta^{-}\}$。

其次，每当一个智能体通过应用规则 Adapt 来动态重分配规划中的智能体时，同时也更新分配差异。另外，一个非空的重分配会产生一个空差异，这时会检测到一次循环。该过程可由规则 Adapt_{cd} 反映：

$$\text{Adapt}_{cd}:\frac{(B,\Upsilon,E,R)\xrightarrow{\text{Adapt}}(B',\Upsilon',E',R')}{(B,\Upsilon,E,R)\rightarrow((B'-\vartheta^{-})+\vartheta^{+},\Upsilon',E',R')}$$

式中 $A^{-}=\{\text{ADiff}(P,d)\mid(p,\tau,z)\in\Upsilon\wedge p\in P\}$；

$A^{+}=\{\text{ADiff}(P,d\circ\Delta(P,B,B'))\mid\text{ADiff}(P,d)\in A^{-}\}$；

$B^{-}=\{\text{Cycles}(P,i)\mid\text{Cycles}(P,i)\in B\wedge\text{ADiff}(P,(\Upsilon^{+},\Upsilon^{-}))\in A^{-}\wedge(\Upsilon^{+}\neq\Phi\vee\Upsilon^{-}\neq\Phi)\wedge\text{ADiff}(P,(\Phi,\Phi))\in A^{+}\}$；

$B^{+}=\{\text{Cycles}(P,j)\mid\text{Cycles}(P,i)\in B^{-}\wedge j=i+1\}$；

$C^{-}=\{\text{Cycles}(P,i)\mid\text{Cycles}(P,i)\in B\wedge\text{Cycles}(P,i)\notin B^{-}\wedge\Delta(P,B,B')\neq(\Phi,\Phi)\}$；

$C^{+}=\{\text{Cycles}(P,0)\mid\text{Cycles}(P,i)\in C^{-}\}$；

$\vartheta^{+}=A^{+}\cup B^{+}\cup C^{+}$；

$\vartheta^{-}=A^{-}\cup B^{-}\cup C^{-}$。

也就是说，只要执行重分配，就要更新相应的分配差异。每次更新都会产生一个空的分配差异，同时增加相应的循环次数。如果一个重分配产生一个非空差异，则循环次数复位为0，这是由于在此情况下循环序列结束，且由于环境的动态性可能会自适应。

最后，每当一个智能体放弃执行规划类型，就应复位分配差异和循环的相关信息。这样只要由 pALICA 表示的规则原始集合中的规则 r 终止执行规划类型，下一条规则 Reset_{cd} 就会复位相应信息：

$$\text{Reset}_{cd}:\frac{(B,\Upsilon,E,R)\xrightarrow{r\in\text{pALICA}}(B',\Upsilon',E',R')}{(B,\Upsilon,E,R)\rightarrow((B'-\vartheta^{-})+\vartheta^{+},\Upsilon',E',R')}$$

式中 $\vartheta^{-}=\{\text{Cycles}(p,i),\text{ADiff}(p,d)\mid P\in\mathcal{P}_{\vee}\wedge\neg(\exists p)(p,\tau,z)\in\Upsilon'\wedge p\in P\}$；

$\vartheta^{+}=\{\text{Cycles}(p,0),\text{ADiff}(p,(\Phi,\Phi))\mid P\in\mathcal{P}_{\vee}\wedge\neg(\exists p)(p,\tau,z)\in\Upsilon'\wedge p\in P\}$。

6.2　冲突消解

在给定上节讨论的检测机制条件下，可制定一个对观测到的冲突的反应。一旦一个智能体检测到 n 次连续循环，则可假设存在一个冲突。为避免冲突，将冲突的规划类型任务分配过程切换到一个集中协调机制。这种局部领导者的概念类似于 STEAM 中的团队领导者[163]，然而在此只是在发生冲突的情况下采用一种基于领导者的决策方法，而不是作为一种默认的处理机制。

最简单的领导者选举方法之一是恃强凌弱（bullying）法[54]。如果自认为是一个领导者时，该方法需要节点来广播选举消息。一个所有可能参与节点范围内的总排序决定了哪个主动节点可成为领导者。因此，每当一个节点从一个优先级更低的节点处接收到消息，就开始广播，如果从一个优先级更高的节点处接收到消息，则停止广播。值得注意的是，在不可靠的异步通信情况下，不能保证只有一个节点认为是领导者。然而在此可假设消息可能正确接收也可能压根没有接收，即在经过通信信道后消息没有失真，以及并不是所有消息都丢失。由于可利用一个合适的误差检测程序来识别并丢弃失真的消息，因此可认为这些假设都是合理的。根据这些假设，一个连续的恃强凌弱算法可保证只有一个节点最终会成为该群中唯一的领导者，该算法中自认为是领导者的节点不会停止广播。

在上述情况中，领导者会定期广播关于规划类型的一个分配。因此可将选举信息与实际情况相结合，从而减少通信开销。在后面的内容中，将这种组合消息称为权威消息，这时所产生的通信协议允许智能体对所接收的权威消息进行快速、及时的反应。一旦接收到权威消息，智能体会有三种操作：首先，如果没有从优先级较高的智能体处接收到权威消息，则该智能体开始广播自身的权威消息；其次，在权威消息在所接收的消息中优先级最高的情况下，将采用该权威消息所包含的分配；第三，如果智能体已知一个具有更高优先级的领导者，则忽略该消息。

当团队分配由领导者决定时，会比正常的分布式决策状态下的反应能力较低。这是由于领导者需要发送其他额外消息以及领导者应被告知其无法观测的变化情况，因此团队应尽量切换到更动态的模式。另一方面，如果团队模式切换回去得太快，冲突源（如信念基不一致）仍存在，导致还会再次发生冲突。因此对规划类型而言，团队处于权威状态的时间间隔应能动态自适应。

一个相应算法见表 6.1。对于每个规划类型，每个智能体都保持在合作模式 Mode(P)，并且利用一个时变间隔 AuthTime(P) 来表示团队处于权威模式的时间。该时间范围为 $t_{\min}$ 到 $t_{\max}$，这取决于冲突检测的频率。可任意选择智能体之间的优先权，但必须符合团队的共同知识。在恃强凌弱算法中通常从一个全序集合

中选择一个唯一的标识符。

将该算法集成到操作语义中非常简单。首先，对于冲突消解，需要额外的信念来反映每个规划类型的状态，因此在信任基中增加下列谓词：

- Mode(P，x）表示规划类型 P 正在模式 x 下执行。
- AuthTime(P，t）表示对于规划类型 P，当前的权威时间为 t。

表 6.1 冲突消解算法

```
foreach(plantype P in execution) do {
 if (Mode(P) = normal) {
    if ( conflict detected or incoming authority message with lower precedence) {
        Mode(P) := authority ;
        t_begin := now;
        AuthTime(P) := min(t_max,AuthTime(P)·t^+);
    }
    if (incoming authority message with higher precedence)
        Mode(P) := commanded;
    else {
        perform reallocation if needed;
        AuthTime(P) := max(t_min,AuthTime(P)·t^-);
    }
 }
 else if (Mode(P) = authority ) {
    perform reallocation if needed;
    broadcast authority message;
    if (AuthTime(P) < t_begin) Mode(P)) := normal;
    if (incoming authority message with higher precedence) Mode(P) := commanded;
 }
 else {
    adopt allocation of last received message with highest precedence;
    if (time since last authority message > t_timeout)
        Mode(P) := normal;
 }
}
```

另外，将下列公理包含在 Σ_b 中：

$$(\forall P \in \text{PlanTypes}^+(z_0))(\exists x)\ \text{Mode}(P, x)$$

$$(\forall P \in \text{PlanTypes}^+(z_0))(\exists t)\ \text{AuthTime}(P, t) \wedge t > 0$$

$$\text{Mode}(P, x) \wedge \text{Mode}(P, y) \rightarrow x = y$$

$$\text{AuthTime}(P, t) \wedge \text{AuthTime}(P, s) \rightarrow t = s$$

$$\text{Mode}(P, x) \rightarrow x = \text{normal} \vee x = \text{commanded} \vee x = \text{authority}$$

前两个谓词是第二个参数的功能，对于每个规划类型，在信念基中该谓词只有一个正值。另外允许执行的模式有正常模式、命令模式和权威模式，这分别表示各自的规划类型是正常执行，或在另一个智能体的控制下执行，或作为领导者执行。

接下来，对所讨论的规则集进行扩展以包括冲突消解。最重要的是，修改 Adapt 规则，使之只能用于不在命令模式下执行的规划类型，即动态重分配只能

工作于正常模式或权威模式下的领导者：

$$\text{Adapt}:\frac{\text{Mode}(P,x)\wedge x\neq \text{commanded}\wedge(B,\Upsilon,E,R)\xrightarrow{\text{Adapt}}(B',\Upsilon',E',R')}{(B,\Upsilon,E,R)\rightarrow((B'-\text{AuthTime}(P,t))+\text{AuthTime}(P,t_n),\Upsilon',E',R')}$$

式中　P——进行重分配的规划类型；

$t_n=\max(t_{\min},\ t\cdot t^-)$。

只要检测到的连续循环次数达到所设阈值 n，就假设该智能体具有权威性。在这种情况下，应切换到权威模式，增大权威时间，并重置循环次数。Auth_{cr}规则反映了该行为：

$$\text{Auth}_{cr}:\frac{(p,\tau,z)\in\Upsilon\wedge p\in P\wedge\text{Mode}(P,\text{normal})\wedge\text{AuthTime}(P,t)\wedge\phi}{(B,\Upsilon,E,R)\rightarrow((B-\vartheta^-)+\vartheta^+,\Upsilon,E,R)}$$

式中　$\phi=\text{Cycles}(P,\ x)\ \wedge x\geqslant n$，$n$ 是一个固定阈值，表示根据定义 6.4 发生的 n 次连续循环；

$\vartheta^-=\{\text{Mode}(P,\ \text{normal}),\ \text{Cycles}(P,\ x),\ \text{AuthTime}(P,\ t)\}$；

$\vartheta^+=\{\text{Mode}(P,\ \text{authority}),\ \text{Cycles}(P,\ 0),\ \text{AuthTime}(P,\ \min\ (t_{\max},\ t\cdot t^+))\}$。

根据权威消息进行冲突消解的核心是 Cmd_{cr}规则，即对接收到的权威消息作出反应。消息接收事件可由 AuthMsg(a', P, C) 表示，其中 a'为发送消息的智能体，P 为规划类型，C 为 a'执行的任务分配。如果接收消息的智能体认为发送消息的智能体具有更高的优先权（$a<a'$），并且也参与执行规划类型 P，则假设将任务分配发送，如果该智能体由此进入一个新的状态，则会针对新进入的状态设计一个任务分配。

$$\text{Cmd}_{cr}:\frac{\text{AuthMsg}(a',P,C)\wedge a<a'\wedge\text{In}(a,p,\tau,z)\wedge p\in P}{(B,\Upsilon,E,R)\rightarrow((B-\vartheta_b^-)+\vartheta_b^++\vartheta_b',(\Upsilon-\vartheta_p^-)\cup\vartheta_p^+,(E-\vartheta_e^-),R)}$$

式中　a——本地智能体；

$\vartheta_b^-=\{\text{Mode}(P,x),\text{Cycles}(P,c)\}\cup\{\text{In}(a'',p',\tau',z')\mid p'\in P\}$；

$\vartheta_b^+=\{\text{Mode}(P,\text{commanded}),\text{Cycles}(P,0)\}\cup C$；

$$\vartheta_b'=\begin{cases}\{\text{Alloc}(z')\} & \text{In}(a,p',\tau',z')\in C\wedge z\neq z'\\ \Phi & \text{其他；}\end{cases}$$

$$\vartheta_p^-=\begin{cases}\{(p,\tau,z)\}\cup\{(p',\tau'',z'')\mid p'\in\text{Plans}^+(z)\} & \text{Alloc}(z')\in\vartheta_b^+\\ \Phi & \text{其他；}\end{cases}$$

$\vartheta_p^+=\{(p',\tau',z')\mid\text{In}(a,p',\tau',z')\in C\}$；

$\vartheta_e^-=\{(b,z_d)\mid(p_d,\tau_d,z_d)\in\vartheta_p^-\wedge b\in\text{Behaviours}(z_d)\}$。

另一方面，如果接收消息的智能体认为自身具有更高优先权，则将切换到权威模式，并根据恃强凌弱算法协议开始广播任务分配：

$$\text{TaskAuth}_{cr}: \frac{\text{AuthMsg}(a',P,C) \wedge a > a' \wedge In(a,p,\tau,z) \wedge p \in P \wedge \text{Mode}(P,\text{normal})}{(B,\Upsilon,E,R) \rightarrow ((B - \vartheta_b^-) + \text{Mode}(P,\text{authority}),\Upsilon,E,R)}$$

式中 $\vartheta_b^- = \text{Mode}(P,\ \text{normal})$。

当达到权威消息的时间阈值，领导者将切换回正常模式，并停止广播。即便没有要求，也会由一个额外消息通知全体团队成员，由此可加速其余队员的模式切换。否则命令智能体将记录缺少权威消息，并根据 DropCmd_{cr}规则经一段延时后切换回正常模式。该时间延迟是非常必要的，因为领导者智能体随时可能损坏而导致整个团队没有领导者。

$$\text{DropAuth}_{cr}: \frac{\text{Mode}(P,\text{authority}) \wedge \text{AuthTime}(P,t) \wedge \phi(P,t)}{(B,\Upsilon,E,R) \rightarrow ((B - \vartheta_b^-) + \text{Mode}(P,\text{normal}),\Upsilon,E,R)}$$

式中 $\vartheta_b^- = \text{Mode}(P,\ \text{authority})$，且如果对于规划类型 P，切换到权威模式的时间大于 t，则满足 $\phi(P,\ t)$。

$$\text{DropCmd}_{cr}: \frac{\text{Mode}(P,\ \text{commanded}) \ \wedge \phi'(P,\ t_{\text{timeout}})}{(B,\ \Upsilon,\ E,\ R) \ \rightarrow \ ((B - \vartheta_b^-) \ + \text{Mode}(P,\ \text{normal}),\ \Upsilon,\ E,\ R)}$$

式中 $\vartheta_b^- = \text{Mode}(P,\ \text{commanded})$，且如果接收到最后权威消息的时间大于 t_{timeout}，则满足 $\phi'(P,\ t_{\text{timeout}})$；

t_{timeout}——常数，与通信频率和期望丢包率有关的参数。

最后，只要本地智能体放弃执行规划类型，就需要重置 $\text{Mode}(P,\ x)$ 所表示的信息。这可由 Reset_{cr}规则表示：

$$\text{Reset}_{cr}: \frac{(B,\ \Upsilon,\ E,\ R) \ \overset{r \in \text{pALICA}}{\rightarrow} \ (B',\ \Upsilon',\ E',\ R')}{(B,\ \Upsilon,\ E,\ R) \ \rightarrow \ (\ (B' - \vartheta^-) \ + \vartheta^+,\ \Upsilon',\ E',\ R')}$$

式中 $\vartheta^- = \{\text{Mode}(P,x) \mid (p,\tau,z) \in \Upsilon \wedge p \in P \wedge \neg(p' \in P \wedge (p',\tau',z') \in \Upsilon')\}$；

$\vartheta^+ = \{\text{Mode}(P,\text{normal}) \mid \text{Mode}(p,x) \in \vartheta^-\}$。

上述规则集通过对某个节点（即规划树中的规划类型）选举一个负责任务分配的领导者，使得团队对冲突具有动态反应和自适应性。最重要的是，领导者的选举只是针对规划树中的某一特定节点，因此规划树中的其余节点不会受冲突消解的影响，而继续以正常动态方式进行。当然也可采用其他机制，例如多数票表决法。这些机制都可以类似的形式集成到 ALICA 中。

显而易见，这种冲突检测和消解的机制满足 5.14 节中所讨论的特性。

命题 15 规则扩展符合规划基公理 Σ_p，并与规划基的树形结构保持一致。

证明：修改规划基的唯一新规则是 Cmd_{cr}。由于唯一的有效分配是由领导者提供，因此该规则采用该有效分配。如果领导者不能提供一个有效分配，则放弃相应规划。在本地智能体是由领导者重分配的情况下，Cmd_{cr}会从规划基中去除当前分支。此时在规划基中最多增加一个元组，并在新进入的状态下进行递归分配。该元组是针对规划类型 P 中的一个规划。由于与规划类型 P 中某一规划相

关的所有元组已去除，因此该命题成立。

命题 16　规则扩展不会产生任何孤立行为。

证明：没有一个新增规则会对执行集增加一个行为。唯一会修改规划基的规则是 Cmd_{cr}，如果该规则从规划基中去除相应的三元组（p，τ，z），则会从执行集中去除所有元组（b，z）。

命题 17　规则扩展满足命题 10。

证明：会修改规划基或形式 In(a，p，τ，z）中的信念的唯一规则是 Cmd_{cr}，该规则以相同方式来更新规划基或信念。

在 10.2 节中将介绍一种对该冲突消解技术的经验评估方法。值得注意的是，不可能用这种方法解决所有冲突。由于不合理规划或由于不允许智能体空闲而不能有效执行规划所导致的冲突不能用该方法解决。利用权威消息解决该问题的方法要求领导者智能体应考虑其未参与的子规划。只要选举出一个领导者，就可通过切换到全局任务分配来实现。在这种情况下，以冲突的规划类型为根节点，领导者可对所有参与智能体计算递归式任务分配。然而如 5.9 节中所述，领导者可能缺乏足够信息来完成任务分配。另一种方法是由在底层规划需求中产生冲突的团队成员来向领导者提供信息。只要智能体不符合权威消息，就可广播其无法执行任务的信息，随后领导者将会考虑该信息。然而由于为了得到一个合适的分配，需要交换可能存在的多个消息，因此该方法可能会由于环境的动态变化而产生一个过时的分配。在今后的工作中，将深入研究该问题。理想情况下，由于冲突消解会在一定程度上降低性能，因此这种冲突是可以完全避免的。实际上，规划的合理性应在一个预运行的有效步骤中验证或由分配产生过程来保证，如通过所采用的规划算法。表 6.2 总结了本章所介绍的信念和规则，通过分配循环来检测冲突以及一个局部领导者来进行冲突消解。

表 6.2　用于表示冲突检测和冲突消解的信念和规则

信念	
ADiff(P，d)	用于区别每个规划类型 P 的当前规划差异。如果由于采用 Adapt 规则而使得差异为空，则产生一次循环
Cycles(P，i)	表示在规划类型 P 的执行过程中，检测到多少次分配循环
Mode(P，x)	x 为规划类型 P 的执行模式。这些模式可以是正常模式、权威模式或命令模式
AuthTime(P，t)	该谓词反映了对每个规划类型 P 的冲突消解的动态自适应持续时间
规则	
$Sense_{cd}$	该规则是对 Sense 规则的改进，用于在信念基中记录分配差异
$Adapt_{cd}$	该规则是对 Adapt 规则的改进，用于更新分配差异和循环周期次数

（续）

规则	
$Reset_{cd}$	该规则对不再执行的规划类型的相关信息进行重置
Adapt	对原始规则 Adapt 进行修改以使得命令智能体不会自主调整任务分配
$Auth_{cr}$	只要一个智能体检测到足够的循环周期次数，就会切换到权威模式
Cmd_{cr}	只要接收到一条来自更高优先级智能体的权威消息，就会将本地智能体切换到命令模式。在这种情况下，强制进行所包含的任务分配
$TakeAuth_{cr}$	用于实现恃强凌弱规则，即只要接收到一条来自更低优先级智能体的权威消息，就会切换到权威模式
$DropAuth_{cr}$	用于一旦达到 AuthTime 表示的持续时间，就强制领导者智能体切换回正常模式
$DropCmd_{cr}$	表示在一定时间内没有接收到权威消息，则智能体从命令模式返回到正常模式
$Reset_{cr}$	不管什么原因，都将重置智能体终止执行的规划类型的模式

第7章 软件架构

尽管软件架构并不是本书的核心内容，但多智能体协作的复杂性和解决方法的多样性值得在此讨论。在此以 ALICA 的参考实现作为讨论的基础。该参考实现可在 BSD 协议下公开获得㊀。

ALICA 引擎是一个执行 ALICA 程序的软件模块。该引擎与规划基及其相关的所有信念、触发行为以及与运行 ALICA 引擎的其他智能体之间的通信均保持一致，为此需要以某种机器可读的形式来表示 ALICA 程序。这可由 7.1 节中介绍的建模工具链来提供，之后在 7.2 节中讨论了 ALICA 参考引擎的内部设计，在 7.3 节将讨论一种可嵌入 ALICA 的整体架构的模型。

7.1 建模工具与交换格式

ALICA 参考实现具有一个称为 ALICA 规划设计器的图形化开发环境，可允许创建所有的 ALICA 语言元素。2008 年 Scharf[144]开发出原始实现，之后进行了不断发展。

ALICA 的实际表示形式是一种基于 XML 的语言，该语言可作为建模工具、执行引擎和模型检验器之间的一种交换格式。这种表示形式是一种利用 Eclipse 建模框架（EMF）[161]实现的特定域语言（DSL）。UML[116]中指定的 EMF 模型会产生可作为编辑器和引擎之间交换格式的 XMI[76]序列化。

规划设计器是一种基于 Eclipse 开发平台[7]的图形化工具。支持对 ALICA 程序中所有部件的建模，即角色、任务、规划、规划类型、效用函数和条件以及在模型中以一种模型驱动开发形式来产生代码。如前所述，建模规划是以 XML 格式进行保存，并随后由运行引擎调用。然而从效率上考虑，该工具还提供了一种用于条件和效用评估的特定平台代码生成机制。由于在运行期间会频繁地进行这些评估，因此生成可直接执行的特定平台代码具有很大的效率优势。为便于直观理解，图形化表示语言元素。

㊀ http：//ros.org/wiki/cn-alica-ros-pkg。

7.2 引擎布局

ALICA 的参考引擎是用 C#语言实现的，并在 mono㊀平台下运行。C#语言是一种强类型的面向对象的命令化语言，通常实时编译 C#程序㊁并采用垃圾收集器㊂。由于这种面向对象的命令化编程方式得到广泛应用并为大家熟知[15]，因此在此采用 C#语言来作为参考实现。另外，与 C + +语言[174]相比，C#语言也更易于阅读和理解，同时仍保留了高效性[56]。最后，还具有其他语言（如 Java）所没有的一些特性，如委托、有限运算符重载、数值类型的泛型、lambda 表达式和 Properties（参见参考文献［24］）。

参考引擎意味着可集成到特定域框架中，该框架是由信念表示框架、通信中间件、感知器和执行器驱动等其他部件组成，由此可编译为一个提供广泛的 API 的库。

图 7.1 给出了一个参考实现的通用布局。其中包括 5 个核心部件和一组附加模块。

引擎接口　引擎接口提供了启动引擎、访问不同部件以及配置运行时行为所需的所有功能。

规划基　规划基是执行一个 ALICA 程序的核心。其中包含程序的运行时表示，即规划结构中智能体的当前状态。在主流程中，主要关注后续访问规划结构的所有相关部件和模块。

规则集　规则集中主要包括规划基定期调用的 ALICA 操作规则，根据这些规则来更新规划结构。

任务分配　任务分配包括了 5.8 节中所讨论的算法。如果需要的话，规则集将应用这些任务分配算法来更新规划结构。

行为池　该部件用于管理智能体所具有的行为集。根据行为规范，在执行集中定期或事件驱动来调用当前行为。同时还提供了行为的所有必要参数。

除了上述的必要部件之外，引擎还负责管理一组重要的可扩展且可替换的模块。

解析器　在给定 XML 文档以及利用 7.1 节讨论的建模工具所编译的库的条件下，该模块对 ALICA 程序的内部表示进行编译。尽管该工作非常必要，但在内存中用其他方法编写程序也是可行的。

同步模块　根据 5.13 节中所定义的规则进行同步传输需要增加某些通信来建立共同信念，这可由同步模块实现。

团队观测器　团队观测器用于处理与团队成员的所有定期通信，需符合程序

㊀ www.mono-project.com。

㊁ 参见 Aycock[4] 和 Arnold 等人[3] 对实时编译与其他技术综述的研究工作。

㊂ 参考文献［79］中深入讨论了垃圾收集器。

中当前状态的内部表示，并根据所接收的消息进行预测。

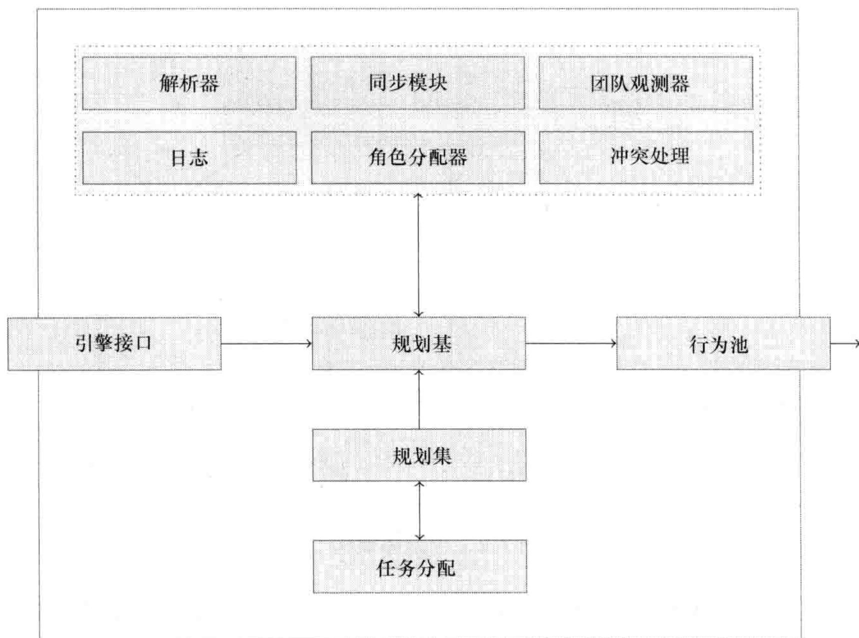


图7.1　ALICA引擎参考架构

角色分配器　根据5.6节所介绍的分配算法，角色分配器为智能体分配角色。

冲突处理　根据第6章内容，冲突处理模块检测并消解任务分配中的冲突。

日志　日志模块提供了一种简单、方便的方式来在文件中记录规则应用情况。

7.3　智能体软件架构

本节将提出一种基于5.2节所介绍的智能体模型的智能体架构。该架构主要是根据Baer[5]的委托，以及参考文献［1］中所提出的特定域架构CANOSA。

图7.2给出了所提架构的布局。直接来自于感知器的信息由一个或多个外部感知部件进行处理。这些信息可用于执行各种不同任务，如图像处理或感知信息融合。将抽象信息发送给环境模型。环境模型是对智能体信念状态的内部处理表示。尽管在某些情况下，并不需要一定要满足环境的内部表示，但在大多数实际场合中，环境以及所需执行的任务都很复杂。因此，信念的内部表示非常有用。在此并不限制如何表示信念，实际上，ALICA引擎并不会直接访问环境模型。因此只要是适用于域的表示方法都可使用，如Bratman[12]的基于BDI的信念模型、FLUX[168]采用的知识表示模型、基于本体的模型（参见参考文献

[25, 41]）或具有最新感知信息的普通数据结构。

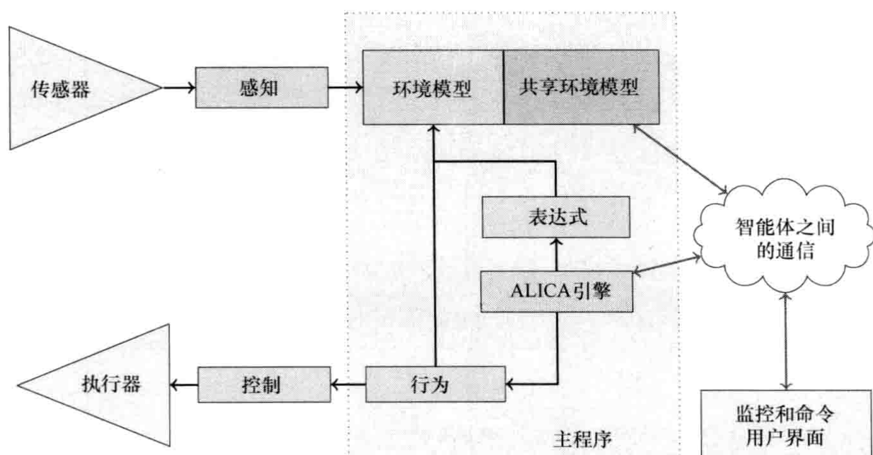


图 7.2 智能体架构

利用一个共享环境模型可扩展环境模型，即集成了来自其他智能体的信息。共享环境模型的任务主要有两方面，首先是估计其他智能体的知识和信念，其次是融合接收信息与本地信息来获得对环境统一且一致的认识。针对该问题，Reichle[131]进行了详细讨论。

如上所述，ALICA 引擎并不直接访问环境模型中的信息，而是通过两个特定域的部件，即表达式和行为。表达式部件包含了规划中所采用的所有公式和效用函数的实现。如 7.1 节所述，这并不是 ALICA 程序基于 XML 的表示，而是独立的程序段，这是因为保持引擎和信念表示之间的松耦合，可能会提高效率。有关智能体内部状态的信念，如 In(a, p, τ, z) 和 Succeeded(a, p, τ) 是由 ALICA 引擎内部管理的，可提供合适的接口来访问这些信念。

行为部件包括 ALICA 程序中所有特定域行为，这些行为由引擎内的行为池控制。反过来，每个行为又与控制部件外部的行为控制器通信，然后这些控制器又与机器人硬件或软件执行器通信。

这种在主程序与附加传感器或执行器相关部件之间的严格划分可易于替换后者。尤其在仿真环境是开发过程一个重要部分的机器人领域，可交换性是非常重要的。

7.4 实现细节

规划基在内部表示为规则集通过遍历可应用规则的一个树形结构。规划基中

的每个三元组对应于树中的一个节点，该节点包含从时间戳到权威消息的各种其他数据结构。利用这种方式，应用规则所需的所有必要信息都可本地获得。

所有语言元素均由唯一的 64 位标识符识别，并保存在哈希图（hash maps）中。由此，为指向规划、任务或状态，智能体只需与这些标识符通信，这样可使得序列化和反序列化的开销最少。

每个行为都通过一个线程来确定，即在执行该行为之前都保持在待机状态，这样可大大减少个体行为的启动时间。引擎内的所有线程都可在固定的时间间隔内相互异步运行，或对引擎接口的信号作出反应。由此，可很容易地通过引擎运行时的行为来适应某个域的具体要求。例如，设置成功信号的行为 Success(b, z) 可立刻激活规划基线程，该线程反过来又依次应用所有规则，由此又可激活另一个行为。由于异步运行线程，因此可消除延迟。

7.5　通信

通过某种智能体间的通信方法，每个智能体在团队内的通信通常基于一个中间件。ALICA 参考实现是基于 Quigley 等人[127]提出的 ROS 系统（机器人操作系统），其中基于一种发布订阅模式提供通信机制。然而由于 ROS 采用一个主进程来建立和管理通信通道，因此目前尚不能应用于多机器人场合，这正是因为该主进程是一个单点故障。为解决该问题，可采用其他中间件系统的代理，如数据分布服务（DDS）[65]。在机器人足球领域，出于效率的考虑，采用一种简单的 UDP 组播代理。

7.5.1　信息交换

pALICA 智能体之间的信息交换可分为两大类：有关环境的特定域信息和 ALICA 内部消息。特定域消息通常反映了发送智能体的全部或部分信念基。这部分的通信不受 ALICA 的控制，假设是通过某些不可靠的通信通道进行通信。

pALICA 定义了下列 3 种通信方式：

- 包括智能体的规划基以及相关的成功信息的状态广播。
- 建立同步传输共同信念的握手协议。
- 包括根据第 6 章对特定规划冲突消解的分配权威消息。

尽管对于多智能体系统，ALICA 是一种域无关的编程语言，但仍主要应用于通常是无线通信方式的移动机器人领域。因此采用一种不可靠的广播媒介进行通信，正如 ALICA 中的消息广播机制。在此假设广播通信的成本等效于点对点通信的成本。

状态广播是以一个动态选择频率 f_{max} 或 f_{min} 定期发送的，发送频率取决于内部状态是否发生变化。状态信息包括智能体的内部状态（即其规划基），以及根据

定义 5.6 已完成的和认为相关的任务—规划二元组。由于智能体不再执行相应规划，因此不会明确发送失败消息。然而为以示区别，成功信息必须明确发送。如果智能体以上述方式接收到失败消息，则可以尝试补偿缺失的智能体，或中止执行相关规划。

这些状态信息的大小取决于是否可从不同的初始状态到达该状态，即规划中包含的状态机是否连通。在非连通情况下，状态可明确地识别相应的任务，这可使得信息大小减半，因此参考实现中假设状态机互不相交。

实际中，由于消息延迟的原因，并不能总是很理想地接收输入的规划基消息。例如，智能体在执行动态重分配之后马上就期望消息到来可能会导致冲突，这是由于消息在环境中检测到相关事件发生之前就已发送。因此应短暂地忽略这种冲突，而延时时间与进行（重）分配后的时间间隔有关。

通过一种三向式握手协议来实现同步协议。如果每个参与智能体都处于状态 z 使得转移 $t=(z, z', \phi)$ 是同步 s 的一个元素并且均认为满足 ϕ，则可声明已准备好同步。每个参与智能体都认可该声明，如果接收到一个来自其他参与智能体的所有声明的确认信号后，就开始广播一个准备好信号。此时支持共同信念，并根据 STrans 规则进行同步转移。一个认为已满足相应条件 ϕ 的智能体一旦接收到一个准备好信号，就会立刻支持共同信念。如果在信念建立期间产生冲突信息，则相应的智能体会通知团队终止执行。

可通过通信建立一种同步方式的共同信念来解决协同攻击问题[62,58]，这是在不可靠的异步通信条件下，已知无确定解的问题，因此任何同步转移协议都只能是近似准确的。认为大多数条件下三向式握手协议已可满足，当然也可很容易地将该基本协议扩展到 n 次握手协议。

一个认为是特定规划类型中领导者的智能体根据第 6 章中介绍的冲突检测和消解机制定期广播分配权威消息。每个消息都要确认发送智能体，以使得可采用恃强凌弱算法，另外每个消息中都包括了对规划类型的全部分配。若领导者智能体不再发送消息，就认为冲突问题已解决。这样团队就切换回正常工作模式，因此该方式也可处理领导者智能体发生损坏的问题。

7.5.2 当前团队估计

对于一个以协调一致方式运行的团队，对团队中参与智能体的集合（即当前团队的组成）进行估计非常重要。尽管 ALICA 假设事先已知整个智能体团队 A，但这并不意味着所有可能参与的智能体都能实际运行，而且智能体还可能随时丧失行为能力。因此每个智能体都需要跟踪上活动团队，该团队定义为参与顶层规划的智能体集合，即 $\{a \mid B \longmapsto \mathrm{In}(a, p_0, \tau_0, z_0)\}$。

若智能体 a 接收到一条来自智能体 b 的消息，则可推断智能体 b 在发送消息时是活动的。假设所有智能体都在可相互通信的范围内，还可根据期望的智能体

个数进行估计，而不是以接收的消息。由于 ALICA 智能体以频率 f_{min} 或 f_{max} 定期广播消息，因此丢失消息的个数在 $t \cdot f_{min}$ 到 $t \cdot f_{max}$ 范围之间，其中 t 为接收到最后一条消息的时间，假设通信抖动可忽略不计。如果 f_{min} 和 f_{max} 之差很小，那么智能体就可估计期望的消息个数为 $\tilde{n}=0.5\ (f_{min}+f_{max})$。

一个特定智能体期望而未接收到的消息个数越多，则原因是消息丢失的概率越小。给定消息发送的概率为 p_m，则由于通信错误所导致消息丢失的概率可大致估计为

$$p_{loss} = (1-p_m)^{\tilde{n}}$$

然而这是一个非常粗略的近似。由于经常出现网络突发错误，因此每个消息的概率相互不独立。智能体依次发送的消息可能会受到同一个突发错误的影响，而且不同智能体同时发送的消息也可能受该错误的影响。

目前针对突发错误及其概率分布已进行了广泛研究。最常用的是 Neyman-A 传播模型[109]，该模型最初用于描述试验田中的幼虫分布。利用该模型，突发错误的长度可用一个泊松分布描述（参见参考文献［44］），由此可将 $\tilde{n}$ 条连续消息的丢失概率模型作为长度为 $k \geqslant \tilde{n}$ 的一个突发错误的概率：

$$p(k \geqslant \tilde{n}) = 1 - e^{-\lambda}\sum_{i=0}^{\tilde{n}-1}\frac{\lambda^i}{i!} \tag{7.1}$$

假设已知机器人损坏的先验概率估计为 $p_{down}(a)$，则每个智能体可由下式来估计整个活动团队：

$$\{a \mid B \longmapsto \mathrm{In}(a, p_0, \tau_0, z_0)\} = \{a \mid p_{down}(a) < p(k \geqslant \tilde{n}(a))\} \tag{7.2}$$

式中　$\tilde{n}(a)$——智能体 a 期望但没有接收到的连续消息的个数。

式（7.2）和 Σ_B 表示如果智能体估计 $p_{down}(a) \geqslant p\ (k \geqslant \tilde{n}\ (a)$，则从信念基的 $\mathrm{In}(a,\ p,\ \tau,\ z)$ 中去除所有信念。由此，例如，若 $p_{down}(a)=10^{-3}$、$f_{max}=15$、$f_{min}=5$ 以及 $\lambda=10$，表示智能体假设另一个智能体 a 在丢失 22 个连续消息之后损坏，即经过 2.2s 没有从 a 接收到任何消息。在机器人足球领域，这些值都是比较合理的假设。

值得注意的是，这种估计没有对不同智能体所接收消息之间的任何依赖关系进行建模。而且当预计智能体会在长时间内进入和超出通信范围时，将不再满足这种简单的近似。另外，除了接收消息或丢失消息，还可能会有其他证据。一个智能体可感知到另一个智能体对环境的影响，并利用该信息推断其内部状态。参见 Huber 和 Durfee[75] 对如何实现无通信下联合意图的讨论。在此，假设有关另一个智能体状态的所有感知信息都可以通信方式表示，或通过行为来直接（如

无线通信连接）或间接对环境产生影响。

在移动机器人场合，机器人会移动超出通信范围，则接收消息的概率是其在环境中位置的一个函数：

$$p_m(\Delta t) = \int_{t_0}^{t_0+\Delta t} f(\mathrm{pos}(a,t),\mathrm{pos}(b,t))\,\mathrm{d}t$$

式中 t_0——最后一条消息的接收时间；

pos（a，t）和 pos(b，t）——在时刻 t 发送智能体和接收智能体的位置；

$p_m(\Delta t)$——在给定发送智能体和接收智能体在环境中的位置的条件下，Δt 期间内成功发送一条消息的概率分布。

如果环境中具有无线电信号或干扰源对象，则概率分布可以是任意复杂的。采用射线跟踪和其他仿真技术可获得该概率分布的精确近似，但在复杂环境中计算量很大（参见参考文献［114，145］）。通常，这种精细模型并不适用于实时性的概率估计。实际中更多采用计算速度较快的方法，这是通过牺牲准确率来提高速度的。通常这些方法仅考虑视距和距离，基于信噪比[148]来估计概率，或利用测试包来测量连接质量[87]。就本书而言，仅采用式（7.1）的简单估计方法。

7.6 本章小结

根据第 6 章所介绍的理论基础，本章阐述了 ALICA 引擎架构。另外，还介绍了用于 ALICA 程序的建模工具和交换格式。尽管 ALICA 引擎可嵌入到任何一种智能体架构中，但在此仅讨论通用架构基础上的必要特性。最后，介绍了参考实现中采用的通信方法以及一种基于期望消息个数来估计团队成员存活性的方式。作为开源代码，参考实现可公开获得。

第 3 部分　通用 ALICA

第 8 章　广义 ALICA

8.1　简介

命题式 ALICA 提供了多种建模选项来处理多智能体情况，然而仅限于命题式语义可导致特定语言元素爆炸式增长，其中最受影响的是行为。由于在 pALICA 中每个行为都是静态的，因此不管在何种情况下调用都基本上会执行相同的动作。尽管可将行为看作黑箱问题而在某些程度上取消该限制，并且因此成为一个与信念基交互作用的图灵完备的程序，但这并没有在 ALICA 中表示，因此不能通过内部 ALICA 的方式进行协调和推理。在此，旨在提出一种行为和规划可参数化的更精细的建模技术以使得可适用于不同情况。由此一种通用的行为 DriveTo(x) 可用于表示 $\text{DriveTo}_{\text{kitchen}}$、$\text{DriveTo}_{\text{Floor}}$和 $\text{DriveTo}_{\text{Door}}$等多种行为。

若需要不确定域的参数，则相应的程序不能再用 pALICA 表示。接下来，将在一些示例域中专门讨论命题式程序的问题。之后提出一种广义的 ALICA，其中规划和行为都可参数化，且只允许在内存范围内进行在参数空间中的推理。

8.1.1　标准情况

首先考虑 RoboCup 场景中的一个问题。在机器人足球中，比赛总是由于犯规而中止，如足球出界或机器人犯规。这种犯规之后应重新是“标准情况”。此时裁判员将足球放置在场上，指定一名进攻方的队员，并命令机器人自定位。经过机器人作出反应并完成定位的一段时间，裁判员重新宣布比赛开始。比赛一旦开始，在允许防守方抢球之后，进攻方需要传球。标准情况中的定位过程是由各种规则来决定的，例如要求防守机器人与足球之间的距离不得小于一定距离（如 3m）。由于经常会发生标准情况，因此能很好定位的团队具有显著优势。由此对于标准情况，团队需花费大量的时间和精力，甚至经常会要求团队针对不同对手或环境条件（如光照不均匀）应有特定的定位能力。在给定不同标准情况（掷界外球、任意球、点球等）以及对手的条件下，可能会有许多不同的策略，这非常类似于实际的足球比赛。

图 8.1 给出了一个 pALICA 中实现定位策略的示意图，其中只有有关该规划中所包含实际策略的少量信息，即只需要一个机器人作为罚球队员来传球，另一个机器人作为接球队员，并安排其他机器人执行防守任务。由于在这三种行为中包含了实际的定位策略，因此对于相同情况的不同策略也非常相似。尽管在 pALICA 中对通过应用规划类型、条件和效用函数可很容易地对相同问题采用和保持几种策略，但如果没有每个策略下几种特殊行为的话，则无法精确表示定位问题的解。这就是本章开始时所提到的命题式爆炸问题。在开发过程中，难以处理和保持大量的必要行为，这可利用广义 ALICA 来避免。

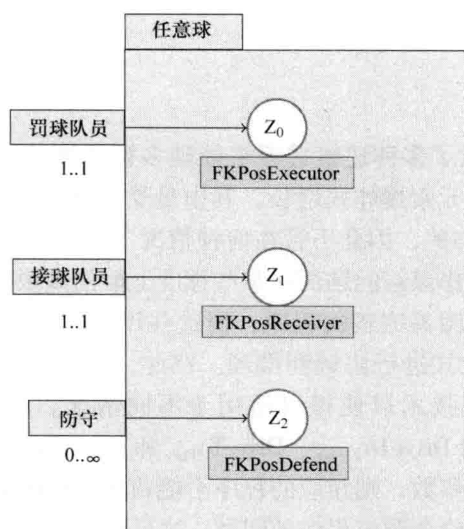


图 8.1　pALICA 中标准情况的规划示例

在此并不详细描述图 8.1 中三种行为的可能实现方法，而只是提炼出应具有的三种共同特性。

相同的执行过程：所有行为都是用于到达球场上一个特定而不同的位置的。给定目标点，采用相同的算法和参数来确定电动机命令（如路径规划和速度控制器）。

复杂的决策过程：由于每个行为的目的不同，因此计算实现过程也不同。实现算法可能会非常复杂，并且会受实现误差的影响。

交互作用：在相互依赖的位置上行为会相互作用。例如，罚球队员、足球和接球队员应构成一条直线，然后沿该直线传球。

在此，认为这些都是普遍特性，同样也会在其他场景中发生，即一些智能体需要在环境中动态协调一阶实体（在该示例中是位置）。

从规划方法和机器学习方法的角度来看，图 8.1 所示的结构也并非理想情况。通常，规划方法和学习方法都是通过搜索一个假设空间［如所有程序空间（参见参考文献［96］）或任务图（参见参考文献［106］）］来对输入问题进行求解的。搜索空间的一个主要特点就是行为问题。对于定位问题，就是要将搜索空间减少到可用行为的各种组合，具有结构少以及用于引导搜索的信息少的特点。

相比之下，一阶方法可大大减少所需行为的个数。在定位问题的特殊情况下，相对于上述提到的常用执行算法，只需一种行为。另外，由于一阶方法不再需要复杂的黑箱，允许规划和学习算法搜索一个表示更加丰富的问题解。

一个可用于该搜索空间的机器学习研究领域是归纳逻辑编程（ILP）。Nienhuys-Cheng 和 Wolf[111] 对 ILP 进行了深入介绍。已有多种方法来解决一阶规划问题，如情景演算中的回归规划算法[134] 或 Kersting 等人[84] 提出的关系贝尔曼算法。在后面的章节中，将通过规划、学习或其他方式生成规划的所有算法都称之为一种生成算法。

8.1.2　积木世界

另一个具有激励性的示例是积木世界（Blocks World），这是最初由 Winograd[179] 提出的一个著名的人工智能玩具问题。在积木世界中，智能体或机器人任务是构建几何图元之外的预定义结构。Nilsson[113] 提出了一种基本的积木世界，其中几何图元限制为立方体积木块和一个容纳所有积木的大平台。每块积木都是一个单一的其他物体（平台本身或另一块积木）。另外，每块积木都是光滑的，或还有一个其他积木。如果机器人不能搬运积木块，则可抓起一个平滑积木，将积木放在平台上或其他积木上。这种基本的积木世界或许是 STRIPS 规划最著名的例子㊀。这甚至会导致某些实际环境中的实验，如 Toussaint 等人[169] 提出的。在后面的章节中，将这种基本的积木简称为积木世界。

一个积木世界问题的解是形式为 Pickup(A)、Put(A，Table)、Pickup(B)、Put(B，A) …的一系列动作。由于 STRIPS 规划是命题式规划，因此这些动作实际上是预先设置好的常数，且由于积木块个数有限，不会构成理论问题。用一个 pALICA 程序来表示该规划非常简单，每个动作都对应于状态中的一个行为，这些状态由通过转移成功完成的相应动作来连接（见图 8.2）。

然而，以这种方式来处理问题可能需要在具有 n 个积木块的积木世界中执行 n^2+n 个行为。产生这么多的行为是非常低效的，尤其是大多数行为几乎相同，

㊀ 列出结合 STRIPS 规划器的积木世界域应用论文、著作和报告已超出本书范畴。在本书撰写时，在 Google 搜索引擎中搜索“条积木世界”关键词可返回 3.2×10^7 个结果。

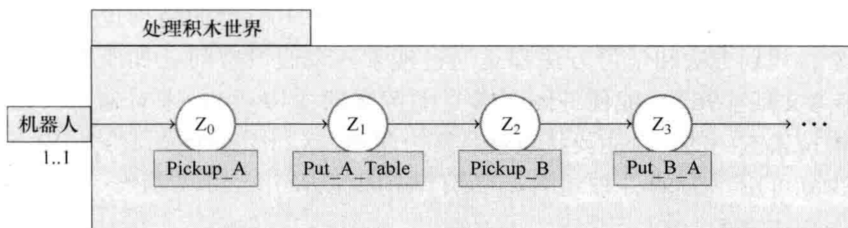


图 8.2 pALICA 中的一个 STRIPS 行为序列

即抓起一个积木块或将其放置在某处。因此执行一个积木世界的规划问题具有与标准情况场景下相同的共同执行特性，然而积木世界方法中没有相互作用，只有一个机器人执行规划，而无需进一步决策，这是由于提供的是一个静态且确定的规划。

接下来，将介绍一种基于本书第 2 部分所讨论的 pALICA 的一种广义 ALICA。表明广义 ALICA 更具有表现力且保持了动态适应情况变化的能力。

8.2 行为参数与规划变量

通过引入类似于 IndiGolog[57] 或 FLUX[167] 等编程语言中一阶动作的参数化行为可解决多种行为共享相同执行模式的问题。

定义 8.1 每个行为 b 都具有一个变量 $\vec{x}$ 的空列表，即行为参数。记 $b(\vec{x})$ 来表示具有参数 $\vec{x}$ 的行为 b。

通过允许行为具有参数，可解决常见的执行模式问题。由此使得程序中一个行为的每次执行都具有一组不同的参数，这样就可以只用两种行为来描述积木世界规划：Pickup(x) 和 Put(x, y)。然而尽管可以用有限个行为来表示无穷多个不同的行为实例㊀，从而只需有限数量的代码，但仍不足以处理运行前值未知或部分已知的参数。这正如上述标准情况场景下的情况。

考虑一个简单示例：一个机器人的任务是在办公室内寻找一本书，并上前确认。在运行之前，由于办公室内有各种不同的书，机器人不知道将要寻找哪本特定书。经过一段时间的搜索并将在信念基中增加相应的信念之后，机器人才会明确应上前查看的具体书。图 8.3 给出了相应的一个规划。机器人不断执行 LookAround 行为，直到在信念基中表示了一本书，并将该书作为参数传递给 Go-

㊀ 或更准确地说，是内存范围内的最多个实例。

to(x) 行为。

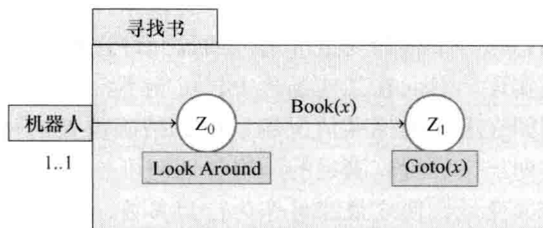


图 8.3　规划示例：寻找一本书

然而，FindBook 规划在 pALICA 中无法表示，即便允许行为具有参数。还需要条件中自由变量和行为参数之间的连接关系。这样就产生了推理任务，在本例中可在规划中定义书的确认。同理，还允许在规划中定义标准情况场景下位置的确认。

接下来，对规划、状态和规划类型（即一个 ALICA 程序中的结构元素）进行概念扩展以使得在规划中具有变量并使其相互关联。

定义 8.2　每个规划 p 具有一个唯一变量 $\vec{x}$ 的空列表，记为 $p(\vec{x})$。每个规划类型 P 具有一个变量 $\vec{x}$ 的空列表，记为 $P(\vec{x})$。

定义 8.3（规划独立性）　所有规划、规划类型和行为都是各自标准化的。

因此，每个规划和规划类型都具有一组由条件指定的唯一变量。根据定义 8.3，不同的规划和规划类型应与两两不相交的变量集相关联。为使得不同规划的变量相互关联，引入绑定的概念。

定义 8.4　每个规划 $p(\vec{x})$ 中的每个状态 s 都定义了一个称为绑定的可能为空的替换 $\theta(s)$，使得 $\theta(s) = \{y_1 \longmapsto x_1, \cdots, y_n \longmapsto x_n\}$，其中，所有 x_i都是相应规划变量中的变量，即 $(\forall x_i)$ var s $(x_i) \subseteq \vec{x}$，而所有 y_i 为 s 中的行为或规划类型变量，即 $(\forall y_i)$ $y_i \in \bigcup_{b(\vec{y}) \in \text{Behaviours}(s) \vee P(\vec{y}) \in \text{PlanTypes}(s)} \vec{y}$。

定义 8.5　每个规划类型 $P(\vec{x})$ 定义了一个称为绑定的可能为空的替换 $\theta(P)$，使得 $\theta(P) = \{y_1 \longmapsto x_1, \cdots, y_n \longmapsto x_n\}$，其中，所有 x_i都是 $P(\vec{x})$ 变量中的变量，即 $(\forall x_i)$ vars $(x_i) \subseteq \vec{x}$，同时所有 y_i 为 P 中的规划变量，即 $(\forall y_i)$ $y_i \in \bigcup_{P(\vec{y}) \in P(\vec{x})} \vec{y}$

由此，每个状态和每个规划类型都声明了规划、规划类型以及所包含行为的一种局部参数化，使得变量相互关联，且允许静态替换。值得注意的是，绑定并不需要绑定子规划或行为中的所有变量，而完全可能不绑定子变量。同理，父规划的变量可经常任意出现在一个绑定中。

通过规划中的条件，规划变量可作为一种动态绑定的表示方式。例如，转移的一个相关条件可以是刚刚发现的一本书，并将该条件与一个规划变量绑定，该规划变量又通过状态与一个行为变量绑定。通过允许在规划变量中含有自由变量的条件可很容易地实现。然而在某些场合中，可能不希望对每个变量都确定一个唯一的下限值。例如在上述的标准情况场景中，精确位置的计算是非常繁琐的，而实际上或许只需列出该位置应满足的一组特性即可。另外，由于可利用绑定来向规划层次结构传递变量，使之最终可作为行为参数，因此可能只需对每个变量值限定为适合于在每层上处理问题的值，而具体细节留给子规划。在这个特定情况下，高层上的规划可描述一般问题的约束，例如机器人必须遵守的比赛规则集，然后可充分利用绑定在一个规划类型中的特定子规划策略。每个子规划又可以根据一个特定策略进一步约束变量，如一个在对方半场罚任意球的风险较大的进攻策略，或一个团队处于领先情况下的防守策略。

该思想启发了可采用约束规划方法，其中每个独立规划不需要传递给行为的基本参数，但必须满足状态要求或约束的基本解。这样多智能体团队的行为可以一种声明方式进行表示，并且与底层求解器的具体实现相解耦。另外，生成算法并不需要关心生成条件是否满足所有情况下传递给行为的所有变量。

FLUX 或 GOLOG[98] 等编程语言要求每个动作在执行时必须完全满足。这是非常重要的，因为在执行相应的动作之前不再进行推理，并且没有定义一个如 Goto(x) 的非基本动作的执行结果。相反在此可允许非基本的约束参数，通过查询所需约束求解器来得到一个合适值，从而通过一个中间推理步骤来求解非基本动作的问题。在 8.4 节中，将正式讨论 ALICA 中的约束公式概念。在第 9 章，将讨论求解约束问题的方法并动态提供基本值。

8.3 智能体变量

尽管规划变量的概念可允许通过一阶项来表现和描述合作，但事实证明，在某些情况下，不足以准确地反映团队的意图行为。

在此考虑 Campbell 和 Wu[21] 所采用的常见的捕食场景。其中，一个智能体团队或智能体群[91] 的任务是搜索和撷取某些物体（如食物或资源），并同时保护基地或巢穴。该场景主要用于检验角色或任务分配技术，以动态决策每个智能体是应去捕食还是保护基地。在该情况下，运行前分配给某个任务的智能体的具体个数未知，只是给定了上限，甚至在运行前智能体总的个数都未知。另外，捕食或保护任务的智能体个数还可动态变化。

在给定一个合适的角色和任务分配方法的条件下，应具体指定每个个体行为。目前，利用 ALICA 中的约束条件来表示这些行为需要规划变量。在给定智

能体的捕食或保护任务下，每个规划变量应分别表示出智能体的目标位置，则变量个数是可能参与的智能体个数的两倍。由于每个智能体在某一时刻只能执行一个任务，因此实际使用的变量最多只有一半，这样在所有情况下的约束问题可能非常复杂。另外，在实际情况下，真正激活的智能体个数总是小于可能激活的智能体个数。

示例 8.1　考虑下面捕食场景中的这个简单示例约束：

$$(\forall a,b \in \mathcal{A})\,\mathrm{MinimalDist}(a,b,\varepsilon) \lor a=b \tag{8.1}$$

$$(\forall a)\,\mathrm{In}(a,P,\mathrm{protecting},z) \rightarrow \mathrm{Near}(a,\mathrm{Base}) \tag{8.2}$$

$$(\forall a,b)\,a \neq b \land \mathrm{In}(a,P,\mathrm{Foraging},\mathrm{Search}) \land \mathrm{In}(b,P,\mathrm{Foraging},\mathrm{Search}) \rightarrow \mathrm{MinimalDist}(a,b,\mathrm{SearchRadius}) \tag{8.3}$$

约束条件表明所有智能体应相互保持最小距离［见式（8.1）］，分配为保护任务的智能体应呆在基地附近［见式（8.2）］，而分配为捕食任务的处于搜索状态下的智能体应相互保持较大距离，以更有效地进行搜索［见式（8.3）］。

特定域谓词 Near 和 MinimalDist 可表示为

$$\mathrm{Near}(a,l) \overset{\mathrm{def}}{=} (\exists p)\,\mathrm{Pos}(a,p) \land \sqrt{(p.x-l.x)^2+(p.y-l.y)^2} < 100$$

$$\mathrm{MinimalDist}(a,b,d) \overset{\mathrm{def}}{=} (\exists p_1,p_2)\ \mathrm{Pos}(a,p_1) \land \mathrm{Pos}(a,p_2)\ \sqrt{(p_1 \cdot x-p_2 \cdot x)^2+(p_1 \cdot y-p_2 \cdot y)^2} > d$$

这种制定约束的简洁方式充分利用了$\mathcal{L}$(Pred，Func）中的特定域谓词和 Pos，并隐含地假设这是第二个参数的函数。这些谓词类似于情景演算中流的函数[134]，但并不表示环境状态或其相关知识，而是表示意图值（预期值），或更确切地说，是允许对意图或责任进行约束。因此，这些约束可看作规划的一种扩展声明，否则以一种程序的方式来形成意图。

为能够规范从在某一特定时间点执行规划或任务的实际智能体中提取的约束，应利用一组相应的公理来弥补约束和信念之间的差距。

定义 8.6　设 F 为二元谓词符号的一个有限集合，对于每个 $P \in F$，设 Σ_A中包含：

$$(\forall a \in \mathcal{A})((\exists x)P(a,x)) \land ((\forall x,y)P(a,x) \land P(a,y) \rightarrow x=y)$$

因此，在给定第一个参数是智能体的条件下，F 中的所有元素都是其第二参数的函数。将 pALICA 的信念公理 Σ_b（定义 5.6）进行扩展以包括 Σ_A。该形式下由信念约束的谓词称为智能体函数流。

由此，Σ_A中的每个公式会保证每个智能体 a 中都存在一个单一值 x，以满足 $P(a,\ x)$。也就是说，$P(a,\ x)$ 会根据名称 P 对智能体 a 附加一个变量。在 Σ_{dom}中对 Σ_A进行特定域扩展可以使得智能体可修改和推理的其他特定域元素具有类似结构。8.4 节将阐述如何将这种形式的约束集成到规划中。

8.4 ALICA 中的约束条件

8.3 节介绍了约束公式的一些示例。直观上，一个约束就是以某种方式限制了域实体有效选择的一个公式。这可由以下定义进行正式描述：

定义 8.7 一个约束是具有自由变量 $\vec{x}=x_1$，…，x_n，$n\geqslant 0$ 的一个公式 $\phi\in\mathcal{L}$。ϕ 称为对 $\vec{x}$ 的约束。每个 x_i 表示域 D_i 中的一个值。对于信念基 B 的一个约束公式的解为一个替代 θ，使得 $\vec{x}\theta$ 为基本解，$\phi\theta$ 与 B 保持一致，且 $\vec{x}\theta\in D_1\times\cdots\times D_n$。

根据定义 5.9，$B\vdash^a_{\mathcal{L},\theta}\phi$ 表示对于逻辑 $\mathcal{L}$ 和 θ 中的信念基 B，智能体 a 证明 ϕ 为一个计算解，这也构成了查询公式 ϕ 的一个解。

定义 8.7 反映了 2.4 节中一个约束满足问题最一般的概念。接下来，为简单起见，避免明确表明域。由于示例 8.1 中包含寻找定量值的变量，因此该示例中的约束并不适合定义 8.7。由于相应的各种排序是有限的，所以这只是一个技术问题[⊖]。因此，示例 8.1 中所用的公式可展开成相应的约束。

定义 8.8 将 ϕ 的展开约束表示为 $\Gamma(\phi)$。$\Gamma(\phi)$ 可通过递归替代由 ϕ 得到：

- 对于某些 p，τ，z，如果 $\mathrm{In}(a,p,\tau,z)$ 或 $\mathrm{Succeeded}(a,p,\tau)$ 在 ψ 中发生，则通过 $\bigwedge_{A\in\mathcal{A}}(\psi\{a\longmapsto A\})$ 所有 $((\forall a)\ \psi)$ 发生。
- 对于某些 p，τ，z，如果 $\mathrm{In}(a,p,\tau,z)$ 或 $\mathrm{Succeeded}(a,p,\tau)$ 在 ψ 中发生，则通过 $\bigvee_{A\in\mathcal{A}}(\psi\{a\longmapsto A\})$ 所有 $((\exists a)\ \psi)$ 发生。
- 如果 $P(A,x)$ 在 ψ 中发生，根据 Σ_A，P 为一个智能体函数流，且 $A\in\mathcal{A}$，则通过 $\psi\{x\longmapsto x_i\}$ 所有 $((\forall x)\ \psi)$ 发生，其中 x_i 是 P 和 A 独有的一个变量。
- 如果 $P(A,x)$ 在 ψ 中发生，根据 Σ_A，P 为一个智能体函数流，且 $A\in\mathcal{A}$，则通过 $\psi\{x\longmapsto x_i\}$ 所有 $((\exists x)\ \psi)$ 发生，其中 x_i 是 P 和 A 独有的一个变量。

在量词消去步骤中，谓词 In 和 Succeeded 的条件用于保证只有智能体范围内的变量是可替换的。显而易见，随着每次操作，不断在公式中去除量词，就会结束约束展开。还要注意，只要已知 $\mathcal{A}$，就可进行约束展开，即在一个 ALICA 智能体的初始化阶段，认为 $\mathcal{A}$ 在整个运行期间都是静止的。

若发生在规划 p 中的公式 ϕ 为针对 p 的本地规划，且 ϕ 中每个变量至多一次作为智能体函数流中的第 2 个参数，则该公式适用于约束展开。值得注意的是，根据标准语义，可认为空的连接等效于 $\top$ 且空的分离等效于 $\bot$。

⊖ 在此没有采用多类逻辑（参见参考文献［19，52］）来定义 ALICA，但原理是相同的。

考虑下列捕食场景中的附加约束：

$$(\forall a,x)\mathrm{In}(a,P,\mathrm{Foraging},\mathrm{Search})\rightarrow \mathrm{Pos}(a,x)\wedge \mathrm{Unexplored}(x) \quad (8.4)$$

直观上，通过利用背景谓词 Unexplored（这可能是在机器人信念状态中表示的某种地图）来约束机器人的位置变量，该约束使得搜索机器人向未探索空间运动。随着机器人的不断运动，之前未探索的区域逐渐已知，因此 Unexplored 也是动态变化的。尤其是当一个捕食机器人运动到已知区域的边界时，地图不断更新，并在每次迭代中，不断探索之前未探索的位置，因此约束满足问题的解也需要不断更新。从语义上，由于一个约束问题的解总是与信念基有关，因此这种对当前信念的依赖关系可由定义 8.7 反映。

ALICA 中的约束意味着控制智能体的行为，因此可存在于规划条件，如作为前置条件的一部分或转移保护条件的一部分。由于约束描述了智能体应处环境的特性，因此与 AgentSpeak[129] 中的目标实现相关。为区别条件和约束，在此引入一种不属于底层语言$\mathcal{L}(\mathrm{Pred},\mathrm{Func})$的公式之间的新二元运算符。

定义 8.9　如果 ϕ 为$\mathcal{L}$中的一个表述，而 ψ 为$\mathcal{L}$中的一个约束，则 $\phi\rightsquigarrow\psi$ 是$\mathcal{L}$中的一个保护性约束。

直观上，$\phi\rightsquigarrow\psi$（称为 ϕ 保护 ψ）所表达的意思是，如果 ϕ 满足，则约束 ψ 也应满足。根据 Frühwirth[49,50] 提出的约束处理规则，该语法和语义都是松耦合的。接下来，将规范化这种概念，并在 8.6 节中讨论运行过程中如何处理保护性约束。

从某种程度上看，这种结构似乎并是不必要的。首先，约束集在连接性（和分离性）下除了表述（或约束）之外是封闭的。其次，如果在一给定约束下无解，则相应的条件应评估为假，然而将约束等效为条件是目光短浅的。正如 2.4 节中所述，取决于$\mathcal{L}$的表示，确定一个解的复杂性可以相差很大，例如从 2SAT[89] 的易处理问题、一般有限域约束的 NP 难题到实数域任意非线性不等式的不确定性问题。另外，尽管找到一个 CSP 问题的解并不比证明存在一个解[27] 更难，但这并不是根据某个目标函数寻找最优解的问题。在 9.6 节中，将介绍 ALICA 中的约束优化，即可通过一个目标函数来比较不同的解。

由此寻找一个约束的解需要花费大量时间，而且即使可能是确定性的一类问题，但求解器也可能为了追求效率而牺牲完备性（如 Selman 等人[147] 提出的 WalkSAT 等局部搜索）。取决于约束编程语言、所用求解器以及所处场景的表现性，对于保护运算符$\rightsquigarrow$可能选择不同的语义。

在此区别关于保护运算符的两种解释，即弱保护解释和严格保护解释。在对于$\mathcal{L}$中的约束求解，以及有限规模约束满足问题的难题容易处理的情况下，应采用一个需要存在解的严格保护解释。在难以求解约束或所采用的求解器不完备时，一个弱解释就更加有意义。

定义 8.10（弱守护解释） 当且仅当满足 ϕ 时，一个保护约束 $\phi \rightsquigarrow \psi$ 在弱解释下才会满足。

因此，在弱解释下，评估保护约束的一个智能体可证明保护条件并声明约束。

定义 8.11（强守护解释） 当且仅当其在弱解释下满足，并对于智能体当前配置中满足 ψ 中所有自由变量的某一替换 θ，满足 $\psi\theta$，则一个保护约束 $\phi \rightsquigarrow \psi$ 才会在强解释下满足。

在强解释下，为评估保护性约束，智能体还需要证明约束解的存在。然而由于约束可能通过自由变量相互关联，因此这样的一个证明仅在由智能体配置提供的背景下才有意义。在 8.5 节中，将讨论这一背景是如何表示的，并在 8.7 节中描述如何由该背景和约束公式产生一个约束满足问题。

当规划中附加有保护约束时，弱守护解释和强守护解释的不同更加明显。在弱守护下，如果智能体评估该守护为真，则进行转移。而在强守护下，智能体还必须证明约束解存在。若两种情况都满足，则约束就会有效，并影响智能体的行为。同理，如果在运行条件下无法得到一个约束解，则必须中止执行相应规划，但只要守护条件满足，参与智能体就会在弱守护解释下继续执行。在这种情况下，即使暂时没有找到解，约束也将继续影响智能体的行为。

理想情况下，只有在守护条件保证约束问题的解存在时，才会形成守护约束，可以很容易地证明，这时强解释和弱解释等效。虽然利用某种包容模型（如 Buntime 的广义 θ 包容[17]）可构建一个对于该守护条件的过于宽泛的近似，但这些方法已超出本书范畴。

为将约束集成到 ALICA 中，相关公式将从$\mathcal{L}$中的表述扩展到守护约束。

定义 8.12 如果 $P(\vec{x})$ 是一个有效的 ALICA 规划，则其前置条件、运行时条件以及转移过程中的所有公式都是守护约束 $\phi \rightsquigarrow \psi$，使得 ϕ 为一个表述，而 ψ 为一个用于展开的公式，这样就将 $\Gamma(\psi)$ 中的所有自由变量都作为智能体函数流或 $\vec{x}$ 之中的第二个参数。另外，前置条件和运行时条件的守护是 p 的局部规划。

后置条件描述了一个行为或规划要实现的目标。一旦某个智能体实现该目标，则中止执行相应的规划或行为，因此后置条件中不会有约束。对行为进行类似扩展可使得前置条件和运行时条件中包含约束。

定义 8.13 如果 $b(\vec{x})$ 为一个有效的 ALICA 行为，则其前置条件和运行时条件为守护约束 $\phi \rightsquigarrow \psi$，使得 ϕ 为一个表述，而 ψ 为一个用于展开的公式，这样就将 $\Gamma(\psi)$ 中的所有自由变量都作为智能体函数流或 $\vec{x}$ 之中的第二个参数。

值得注意的是，行为仍嵌入在规范行为规划中，如 5.7 节所述。

8.5　约束条件库

约束逻辑规划系统通常具有作为一个约束库的特点，即在操作层上可看作一个具有并执行约束的库。一旦出现新的信息，则“激活”相关约束，并根据约束处理规则来传递信息。一旦发现所存储的约束集合不一致，则回溯系统[2]。

从一个更加声明性的角度来看，约束库可看作广义的经典替换，这是由于表示可能估值信息的约束以及任一替换都可表示为一组约束。

在 ALICA 中，每个智能体都具有一个约束库，该约束库中包含了一组智能体认为目前正激活的约束。由于约束都是存在于规划或行为中，因此约束库中的每个约束都是由相应的规划或行为注解。

定义 8.14　一个约束库是一个可能为空的元组（P，ϕ）集，其中 P 是一组规划，而 ϕ 是一个约束。

在经典的约束规划系统中，约束库由简化规则或传播规则进行修改。例如，删除一个在所有情况下都满足的约束，或两个约束合并为一个约束。在 ALICA 中，随着智能体遍历规划树，可从约束库中增加或去除约束。因此在每个时间点，智能体都必须能够去除之前所声明的约束，这可通过由规划集对每个约束进行注解来实现。一个由规划 p 声称的约束 ϕ 可作为（$\{p\}$，ϕ）增加到约束库中，这就可以在给定规划的条件下来确定约束。在基本情况下，约束集只包括一个规划。在 8.6.2 节中，将介绍一个根据 Frühwirth[50] 来反映约束传播的操作规则，可得到一个具有多个元素的集合。

在此，将定义 5.1 中的智能体配置概念进行扩展以包含一个约束库。

定义 8.15（智能体配置）　一个 ALICA 中的智能体配置是一个元组（B，Υ，E，R，G），使得：

- （B，Υ，E，R）是 pALICA 中的一个智能体配置；
- G 是一个约束库。

对于任意一个智能体 $a \in \mathcal{A}$，Conf（a）表示该智能体的配置。初始的智能体配置包括一个空约束库，即形式为（B，Φ，Φ，Φ，Φ），使得 B 不对团队状态进行任何假设。尤其是，其中不包含任何形式 In（a，p，τ，z）的信念。

8.6　规则

类似于 pALCIA，一个 ALICA 智能体的内部状态可根据一组操作规则进行转移。与 pALICA 不同的是，这些规则还与一个包含一组有效约束或意图的约束库有关。在 8.6.1 节，将 5.13 节中介绍的原始规则系统扩展到一般情况，并描述

约束库如何以及在何时进行更新。之后在 8.6.2 节中，讨论一种根据约束处理规则[50]来更新约束的扩展。

8.6.1 提升命题式 ALICA 的规则

5.13 节中描述的规则系统大体上可直接应用到通用 ALICA。在此仅需强调一个智能体配置的新元素，即包含可影响当前智能体行为的有效约束集合的约束库。接下来，对于相应的 pALICA 规则，定义每个 ALICA 规则。其中大部分规则的变化都很小，只是受约束库的影响。在此记

$$T \in S: \frac{(B, \Upsilon, E, R) \xrightarrow{T} (B', \Upsilon', E', R')}{(B, \Upsilon, E, R, G) \rightarrow (B', \Upsilon', E', R', G')}$$

来表示如果应用 pALICA 规则 $T \in S$ 将 (B, Υ, E, R) 更新为 (B', Υ', E', R')，则原始配置为 (B, Υ, E, R, G) 的一个智能体也将更新配置为 $(B', \Upsilon', E', R', G')$。对于大多数规则，已足以撤销不再执行的规划约束。在此将一个限于规划基Υ的约束库 G 表示为 $C(G, \Upsilon)$：

$$C(G, \Upsilon) \stackrel{\text{def}}{=} \{(S, \psi) \mid (S, \psi) \in G \wedge (\forall p \in S)(p, \tau, z) \in \Upsilon\}$$

在利用规则执行一个规划，必须声明相应的前置条件和运行时条件的约束的其他情况下。为表示简单起见，利用一个宏 $D(G, \Upsilon)$ 来表示撤销有关不再执行的规划的所有约束，并声明Υ中有关规划前置条件和运行时条件的所有约束：

$$D(G,\Upsilon) \stackrel{\text{def}}{=} C(G,\Upsilon) \cup \{(\{p\},\psi) \mid (p,\tau,z) \in \Upsilon \wedge \text{Pre}(p) = (\phi \rightsquigarrow \psi) \vee \text{Run}(p) = (\phi \rightsquigarrow \psi)\}$$

对约束库无影响的规则： 规则 Sense、BSuccess、TSuccess、BAbort、BRedo、BProp 和 RoleAlloc 以及除了 Cmd_{cr}之外的冲突检测和消解的所有规则都不会改变规划基，因此也不会改变约束库：

$$T \in S: \frac{(B, \Upsilon, E, R) \xrightarrow{T} (B', \Upsilon', E', R')}{(B, \Upsilon, E, R, G) \rightarrow (B', \Upsilon', E', R', G')}$$

式中 $S = \{\text{Sense}, \text{BSuccess}, \text{TSuccess}, \text{BAbort}, \text{BRedo}, \text{BProp}, \text{RoleAlloc}, \text{Sense}_{cd}, \text{Adapt}_{cd}, \text{Reset}_{cd}, \text{Auth}_{cr}, \text{TakeAuth}_{cr}, \text{DropAuth}_{cr}, \text{DropCmd}_{cr}, \text{Reset}_{cr}\}$。

限制约束库的规则： 规则 PAbort、PRedo、PReplace、PProp 和 NExpand 会从规划基中去除三元组，因此这些规则会扩展有效规划约束的约束库：

$$T \in S: \frac{(B, \Upsilon, E, R) \xrightarrow{T} (B', \Upsilon', E', R')}{(B, \Upsilon, E, R, G) \rightarrow (B', \Upsilon', E', R', C(G, \Upsilon'))}$$

式中 $S = \{\text{PAbort}, \text{PRedo}, \text{PReplace}, \text{PProp}, \text{NExpand}\}$。

声明和撤销约束的规则： 规则 Alloc、Adapt 和 Cmd_{cr}会通过增加和去除三元组来改变规划基。这可通过仅针对仍在执行规划的限制约束，以及新执行规划的

前置条件和运行时条件的声明约束来影响约束库。值得注意的是，这三条规则仅增加初始状态的三元组。

$$T \in S: \frac{(B, \Upsilon, E, R) \xrightarrow{T} (B', \Upsilon', E', R')}{(B, \Upsilon, E, R, G) \rightarrow (B', \Upsilon', E', R', D(G, \Upsilon'))}$$

式中　$S = \{\text{Alloc}, \text{Adapt}, \text{Cmd}_{cr}\}$

清除约束库的规则：通过规则 Init，一个智能体可初始化其配置。在发生失败或向顶层规划传播失败的情况下，可利用规则 PTopFail 来处理失败，并触发一个重初始化。在上述两种情况下，清空约束库，并对智能体清除所有之前表示为约束的意图。

$$T \in \{\text{Init}, \text{PTopFail}\}: \frac{(B, \Upsilon, E, R) \xrightarrow{T} (B', \Upsilon', E', R')}{(B, \Upsilon, E, R, G) \rightarrow (B', \Upsilon', E', R', \Phi)}$$

现在只剩下处理转移的两种特殊规则。

转移规则：利用转移规则，智能体可在状态机中实现一个转移，由此产生相应的声明约束：

$$Trans: \frac{(B,\Upsilon,E,R) \xrightarrow{\text{Trans}} (B',\Upsilon',E',R')}{(B,\Upsilon,E,R,G) \rightarrow (B',\Upsilon',E',R',C(G,\Upsilon') \cup \{(\{p\},\psi)\})}$$

式中　ψ——智能体发生转移（z_1，z_2，$\phi \rightsquigarrow \psi$）的约束；

p——包含该转移的规划。

由此，随着一个进入新状态的转移，智能体会在约束库中增加相应的约束。值得注意的是，由于任何一个表述 ϕ 都可重记为一个等价的守护约束 $\phi \rightsquigarrow \top$，因此约束可表示为$\top$。另外，去除不再执行的规划的所有约束。

同步转移规则：与转移规则类似，同步转移规则也是在约束库中增加约束，并去除智能体不再执行规划的约束。然而由于在同步行为中涉及多个智能体，则应增加同步中每个转移的约束：

$$\text{STrans}: \frac{(B, \Upsilon, E, R) \xrightarrow{\text{STrans}} (B', \Upsilon', E', R')}{(B, \Upsilon, E, R, G) \rightarrow (B', \Upsilon', E', R', C(G, \Upsilon') \cup F)}$$

由此使得若 s 为所考虑的同步行为，p 为 s 发生时的规划，则 $F = \{(\{p\},\psi) \mid (z_1,z_2,\phi \rightsquigarrow \psi) \in s\}$。

将规则集集成到约束库的扩展与 5.14 节中所讨论的特性保持一致，即规划基仍符合规划基公理，没有引入其他孤立行为，并且智能体仍信任其所执行的意图行为。这是由于对规则的提升只是产生了用于处理约束库的规则，而对智能体配置中其他元素的处理仍保持不变。

8.6.2　约束处理规则

Frühwirth[50] 定义了一种基于约束处理规则（CHR）的声明性编程系统。尽

管这些规则并不是本书工作的重点，但也可能会增强一个具有约束处理规则的ALICA引擎来简化随后对求解特定变量查询等推理步骤。目前尚未在具体实现中得到应用，在此简单描述如何形成这种扩展。

由 Frühwirth[50] 所给出的下列定义反映了CHR编程的语法。

定义 8.16　一个CHR程序是一个CHR的有限集，每个CHR都是下列形式之一：

- 一种简化CHR的形式为：$H_1, \cdots, H_i \rightarrow G_1, \cdots, G_j \mid B_1, \cdots, B_k$。
- 一种传播CHR的形式为：$H_1, \cdots, H_i \rightarrow G_1, \cdots, G_j \mid B_1, \cdots, B_k$。
- 一种简化传播CHR的形式为：$H_1, \cdots, H_l \setminus H_{l+1}, \cdots, H_i \rightarrow G_1, \cdots, G_j \mid B_1, \cdots, B_k$。

其中，$i>0$，$j \geqslant 0$，$k \geqslant 0$，$l>0$，且多个 $H_1, \cdots, H_i$ 为CHR约束的一个非空序列，守护 $G_1, \cdots, G_j$ 是一个内建约束的序列，$B_1, \cdots, B_k$ 是一个内建约束和CHR约束的序列。

在CHR框架中，内建约束是完全由CHR定义的更复杂约束的基本构建块。

直观上，如果对于一条简化规则中的每个约束头 H_x，在约束库中都存在一个实例 $H_x\sigma$，且可证明规则的相应守护实例 $G_1\sigma, \cdots, G_j\sigma$ 具有答案 θ，使得 θ 不能代替规则头中的变量，则所有 $H_x\sigma$ 都可从约束库中去除，并由约束体 $B_x\sigma\theta$ 替代。传播规则相对简单，但不能从约束库中去除约束头。简化传播规则是简化和传播两者的结合，会去除 $H_1\sigma, \cdots, H_l\sigma$，但不能去除 $H_{l+1}\sigma, \cdots H_i\sigma$。所得到的这种系统是图灵完备（Turing complete）[160]的。

在此，可提供一种将传播CHR集成到ALICA中的操作规则。与原始CHR系统不同，不再从约束库中去除ALICA中的一个约束，直到智能体不再执行相应的规划。给定一组作为ALICA的CHR扩展的传播规则，下列规则CProp定义了对智能体配置的相应更新：

$$\text{CProp:}\ \frac{(B \vdash_\theta (G_1 \wedge \cdots \wedge G_j)\ \sigma) \wedge (S_1,\ H_1\sigma) \in G \wedge \cdots \wedge (S_i,\ H_i\sigma) \in G}{(B,\ \Upsilon,\ E,\ R,\ G) \rightarrow (B,\ \Upsilon,\ E,\ R,\ G')}$$

式中　$H_1, \cdots, H_i \rightarrow G_1, \cdots, G_j \mid B_1, \cdots, B_k$——CHR扩展中的一条传播规则；

θ——守护条件证明的计算解；

$G' = G \cup \{(T,\ B_i\sigma\theta) \mid 1 \leqslant i \leqslant k\}$；

$T \in \cup_{k=1}^{i} S_i$。

如果对于当前信念集满足守护条件，且一个规则头实例有效（即在约束库中），则规则CProp将声明在约束库中增加一条传播规则体。类似于在CHR编程语言中的规则应用，来为避免没有中止，对同一约束应至少应用一次CRrop规则。利用该规则，就可能传递部分CHR语义工作于ALICA。

由于约束库是动态的且在任意时刻反映有效约束集，因此不可能以上述方式

来嵌入简化规则。考虑两个元组（$\{p_1\}$，ϕ_1）和（$\{p_2\}$，ϕ_2）是约束库中一部分的一个实例。应用简化规则可去除这两个元组，并增加一个相应简化规则体的元组，如（$\{p_1, p_2\}$，ψ）。如果智能体现在不执行 p_2，则必须从约束库中撤销这一结果，因为 p_2 的约束不再相关。因此，智能体还要撤销可能仍有效的原始约束 ϕ_1。

8.7　查询

到目前为止，约束都是在一个约束库中维护和管理的，但这并不会影响智能体的行为。通过查询可实现上述要求。在智能体、行为以及其他部件的运行过程中都可查询约束库。

在此，来区分只需证明存在解的存在性查询和需要具体解的具体查询这两类查询。如上所述，尽管这些问题相当复杂，但一旦利用了 9.6 节中的约束优化，这就不再是难题，因此需要进行区分。

在强守护解释下（见定义 8.11），评估规划—条件时，操作规则可进行存在性查询。

定义 8.17　一个查询 q 是一个由规划或行为的查询内容 c、可计算值的变量列表 $\vec{v}$ 和 $\vec{v}$ 中自由变量的约束公式 ϕ 组成的三元组（c, $\vec{v}$, ϕ）。

为了响应查询，对于当前的智能体配置（B, Υ, E, R, G），可构建一个相应的约束满足问题。直观上，这可通过从向顶层规划提出查询的内容中向上遍历规划树，收集所有有效约束并在遍历过程中绑定来实现。根据局部性原理，不考虑在子节点和旁支节点中有效的约束，由此规划 p 提出的查询或许比其父规划提出的同样查询更加具体。

图 8.4 中的递归函数 t 可反映规划树的向上遍历过程。第一种情况是达到顶层规划时的终止条件。第二种情况是相关性，即若查询内容 c 为一个行为或一个规划类型，则利用父状态的绑定 $\theta(z)$ 并增加与相应父规划 p 关联的所有有效约束。最后一种情况是利用相应规划类型 P 的绑定来处理规划。

利用这种方式，用于查询（c, $\vec{v}$, ϕ）的遍历 $t(c, \vec{v}, \phi, \Phi, \Upsilon, G)$ 就可产生一个由要求解的 CSP ψ 和所有绑定组合的替换 θ 组成的二元组（ψ, θ）。所以，CSP 可构建为 ϕ 和约束库 G 中所有相关有效约束的结合。

替换 θ 反映了查询变量与查询内容之间的关系。由于绑定了状态和规划类型之间的传递，每个变量都可由 θ 映射到规划树中发生概率最高的规划。这种替换对解的跟踪和协调具有重要作用，这将在 9.5 节介绍。

随后，智能体可证明 $B \vdash_\eta \psi$ 是对查询的响应。在一个具体查询情况下，$\vec{v}\theta\eta$ 就是所得的解矢量。

$$
t(c, X, \phi, \theta, \Upsilon, G) = \begin{cases}
(\phi, \theta) & c = p_0 \\
t(p, X', \phi', \theta', \Upsilon, G) & (p, \tau, z) \in \Upsilon \wedge (c \in \text{Behaviours}(z) \\
 & \quad \vee c \in \text{PlanTypes}(z)); \\
 & \text{式中 } X' = X\theta(z) \\
 & \quad \phi' = \phi\theta(z) \wedge \\
 & \quad \wedge (\exists \mathcal{P}, v)(\mathcal{P}, \phi) \in G \wedge p \in \mathcal{P} \wedge v \in \text{vars}(\Psi) \cap X'\Psi; \\
 & \quad \theta' = \theta \circ \theta(z)。 \\
t(P, X', \phi', \theta', \Upsilon, G) & (c, \tau, z') \in \Upsilon \wedge P \in \text{PlanTypes}(z) \wedge \\
 & \quad c \in P \\
 & \text{式中 } X' = X\theta(P); \\
 & \quad \phi' = \phi\theta(P); \\
 & \quad \theta' = \theta \circ \theta(P)。
\end{cases}
$$

图 8.4　CSP 构建：规划树的自下而上遍历

示例8.2　对于一个用于在 RoboCup 场景中进攻的简单策略，带球的机器人朝可得分的位置运行。第二个机器人试图阻止对手接近球，其余两个机器人位于靠近本方球门处，这样万一对手得球，可进行防守。图 8.5 给出了一个相应的 ALICA 规划。

规划由三个相应的任务组成，一旦进攻队员朝对方球门射门（不管是否得分），就认为已成功完成。其他机器人的具体行为在这个层面上不会体现，而分别由规划类型 Protect(x) 和 DefendGoal 封装。

规划语义在很大程度上是由其运行时条件 ϕ 决定的，该运行时条件可定义为

$$
\begin{aligned}
\phi = \phi' \rightsquigarrow \psi = & ((\exists a, z)\,\text{In}(a, \text{Attack}, \text{Attacker}, z) \wedge \text{HasBall}(a)) \rightsquigarrow \\
& \Gamma(\text{ScorePosition}(x) \wedge ((\forall a)\,\text{In}(a, \text{Attack}, \text{Defender}, z_4) \rightarrow \\
& \quad (\exists \text{defPos})\,\text{TargetPos}(\text{a}, \text{defPos}) \wedge \\
& \quad \text{Between}(\text{OwnGoal}, \text{Ball}, \text{defPos})))
\end{aligned}
$$

因此，如果执行进攻任务的机器人没有控球（HasBall(a)），则必须放弃执行该规划。另外，在 ϕ 中为自由变量的规划变量 x 表明了进攻机器人带球所朝向的位置。x 是由宏 ScorePosition(x) 进行约束，这个宏的具体细节并不重要，但应考虑对手位置以及机器人踢球装置的物理限制。除此之外，ϕ 还限制了位于球

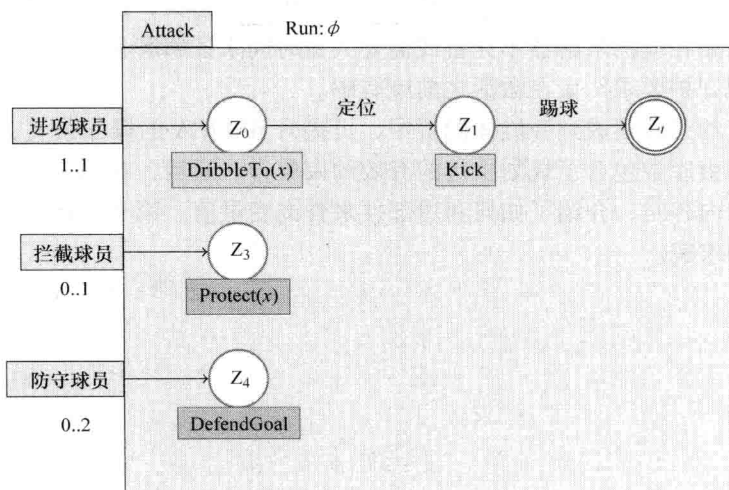


图 8.5 规划示例：RoboCup 中的进攻策略

与球门之间的所有防守机器人的智能体函数流 TargetPos。关于拦截队员和防守队员位置的具体细节留给读者。

在每次迭代过程中，进攻队员执行的 DribbleTo(x) 将进行一个查询（DribbleTo，(x)，$\top$）来获得一个目标位置，该位置又转换为一个运动指令。遍历整个规划树可产生一个对规划类型 Protect(x) 或 DefendGoal 没有任何约束的 CSP $\top \vee \psi$。直观上，其具体细节与进攻队员无关，该 CSP 已足以得知防守队员位于其身后。

另一方面，拦截队员执行可实现根据进攻队员目标位置 x 来约束拦截位置的规划的规划类型 Protect(x)，通过遍历，对 Protect(x) 中拦截位置的所有查询增加 ϕ 的约束。同理，在规划类型 DefendGoal 中也存在另一组用于确定防守队员位置的约束。在 DefendGoal 中通过位置查询所构建的 CSP 中包括 ψ，由于 ψ 表明了 CSP 的需求，即根据智能体函数流 TargetPos 对查询共享变量。

确定智能体实际位置与其约束目标位置之差非常重要。对于某一给定时刻，实际位置是固定且不能改变的，而由智能体函数流 TargetPos(a, p) 反映的目标位置要受到任意约束的影响。

8.8 本章小结

由于许多问题无法利用命题式 ALICA 来很容易地表示，因此利用规划、规划类型、行为和智能体的变量对该基本语言进行扩展，这允许约束作为部分条件

集成到编程语言中。在此讨论了两种不同的守护约束语义，即弱守护语义和强守护语义。只要相应的求解器不完备或需要大量时间来证明解存在时，就应选择弱守护。而强守护要求约束在激活之前应有解。

通过将约束库集成到智能体配置中，可提升 pALICA 的操作语义，使其更具表现力。约束库中包含了规划执行的有效约束集合。最后，在给定一个规划或行为作为查询内容时，介绍了如何利用部件来查询变量值。第 9 章将重点介绍约束满足问题的求解。

第 9 章　约束问题求解

至此，已讨论了一个 ALICA 智能体如何进行约束管理。约束管理和更新的方式对所采用的求解器没有任何限制，根据所形成的约束问题类可选择相应的求解器。然而讨论所有可能或甚至合理的约束问题类的求解器已超出本书范畴。接下来，将着重考虑充分表达各种机器人领域的特定约束满足问题类。首先，研究机器人领域中某些示例域的表示需求。根据这些示例，正式定义一个面向机器人和多机器人领域的约束问题类。之后，针对该问题类提出一种求解器，并对其进行扩展以使得可求解约束优化问题、随时间变化跟踪解，以及在团队中协调解。

9.1　典型约束满足问题

再次考虑机器人足球领域的标准情况。在该标准情况下，若给定球的位置以及所有已感知对手的位置，最多需要确定四个机器人在场上的位置。大多数情况下，这种问题具有 12 维（由于由一个二维空间点和朝向来表示一个位置，则每个机器人有三维）。与工业领域中的典型约束满足问题相比，这是非常小的[㊀]。然而正如 9.4 节中将要讨论的，在软实时条件下需要快速求解。另外由于这是一个动态域，因此解会随时间变化。

在标准情况下，描述行为所需的特定约束通常可分为以下三类：

- 线性：使得机器人位于矩形区域之内或之外。
- 二次型：在涉及机器人与目标之间的距离时。
- 多项式：在需要利用非常复杂的几何关系时，如周长测试。

另外，由于可描述多个不同解，这些约束问题并不是简单地连接。例如，考虑一个限定两个机器人位置的任意球标准情况，使得一个机器人（p_1）接球，而另一个机器人（p_2）阻止对手断球。这样，一个直观的团队指令可描述为："如果有一个对手能够断球，则应由 p_2 进行拦截"。为阅读方便，用两个变量 o_x 和 o_y 来表示目标 o 的位置。在后面的章节中，将球记为 b。

如果相对于拦截点与球之间的距离，对手 o 与传球矢量 $\vec{d}=p_1-b$ 更近，则假设该对手可断球，记为

㊀ 例如，与可满足性模理论竞争（SMT-COMP）中的基准问题相比。

$$\vec{c} = (o - b)$$

$$\vec{v} = \frac{1}{|\vec{d}|}\begin{pmatrix} c_x d_x + c_y d_y \\ c_x d_y - c_y d_x \end{pmatrix}$$

这可由下式表示：

$$\phi = v_x > 0 \land v_x < 1 \land v_x^2 > v_y^2$$

因此，如果对手满足 ϕ，则拦截队员应位于对手之前：

$$\psi = |p_2 - v_y \vec{d}_\perp + o| < 0.05$$

式中 $\vec{d}_\perp$——与 $\vec{d}$ 正交的矢量。

由于足球比赛中会有多个对手，因此要对每个感知到的对手在约束问题中相应地增加一个变型的 $\phi \to \psi$。

解决这种约束满足问题的一种方法是将每个变量的域离散化为一个合适的栅格（如 1cm 宽），并求解所产生的有限域问题。然而这会产生一个非常大的解空间，尤其是需要考虑多个机器人的位置时。在这种特定情况下，整个空间可能包含大约 4.7×10^{12} 个元素（有四个机器人时，则为 2.2×10^{25} 个元素）。

这种指数效应可轻易地抵消有限域求解器对连续域求解器的优势，而且由于失真及其他效应，离散化还可能导致特定约束条件的数值计算问题。因此对于这种情况，认为实数上的有理不等式布尔组合可构成最适合的约束系统。

在另一个场景中，假设一个移动机器人的任务是抓取一个物体，如从厨房取一杯咖啡。在机器人上需要配置一个机械臂，利用其末端执行器可抓取杯子大小的物体。为完成该任务，机器人具有四种行为，即 DriveTo($\vec{x}$)、PositionGripper($\vec{y}$)、Grab 和 Release。DriveTo($\vec{x}$) 行为可在建筑物中导航以到达由直角坐标 x 和 y 以及朝向 α 组成的位置（记为 $\vec{x}$）。PositionGripper($\vec{y}$) 行为是用于控制机械臂，使其末端执行器达到三维矢量 $\vec{y}$ 所描述的位置。为简单起见，忽略指定末端执行器的朝向。另外，假设机械臂的原点位于机器人中心，具有 5 个转动关节。设 M_i 表示描述第 i 个关节相对于其角度 β_i 的转换矩阵，则末端执行器在世界坐标系统中的位置可由下式计算：

$$\vec{y} = M_5 M_4 M_3 M_2 M_1 M_0 \ (x,\ y,\ 0,\ 1)^{\mathrm{T}}$$

式中 M_0——相对于机器人朝向所产生的旋转。

除了标准情况的场景之外，逆运动学问题还需要三角函数，如正弦函数和余弦函数。9.2 节将构成一个适合于表示上述问题的约束问题类。

该方程具有 11 个变量，每个自由度一个变量，再加上末端执行器的位置变量和机器人的位置变量。因此，确定 $\vec{x}$ 和 $\vec{y}$ 的两个问题相互关联，再结合厨房中杯子的位置以及厨房结构布局的相关信息，就可构成一个简单的约束问题，使得机器人所有行为都以某种方式进行交互，即机器人应运动到能抓取杯子的恰当

位置。

9.2 非线性连续约束满足问题

在参考文献［155］中，以下列方式讨论了非线性连续约束满足问题的一类问题：

定义9.1 一个非线性连续约束满足问题（CNLCSP）是一个由下列元素组成的三元组（ϕ，X，C）：

- 具有变量 P 的一个命题公式 ϕ。
- $\mathbb{R}$域内的 n 个变量集 X。
- 每个规划 $p_i \in P$ 确定一个约束 $c_i \in C$，使得 $c_i = f_i(\vec{x})\ o_i 0$，其中 $o_i \in \{<, >, \leqslant, \geqslant, =, \neq\}$，$\vec{x} \subseteq X$，并且所有 f_i 都是 $\mathbb{R}^k \longmapsto \mathbb{R}$ 的任意函数。

一个 CNLCSP 可理解为一个价值函数 v：$X \longmapsto \mathbb{R}$，通过

$$v(p_i) = \begin{cases} \top & v(f_i(\vec{x}))\ o_i 0) \\ \bot & \text{其他} \end{cases}$$

可扩展为 $P \longmapsto \{\top, \bot\}$。

当且仅当 ϕ 中所有变量 p_i 都可由在经典命题式 CNLCSP 中评估为$\top$的 $v(p_i)$ 替代时，v 可理解为公式 ϕ 的解。

另外，在参考文献［155］中，对处理来自于完全不同研究领域约束问题类的求解器进行了评估，如区间传播与分裂（参考文献［2，171］）、可满足性模理论（SMT）（参考文献［6，48］）以及约束满足问题所定义的场景局部数值搜索。SMT 是一种求解结合特定求解理论与布尔满足性（SAT）求解器的不同理论问题的新方法，因此根据这种理论的求解器可对更具表现力的约束问题类具有 SAT 求解器（如参考文献［102］）的某些速度特性。

然而在评估过程中，局部数值搜索方法在对源自机器人场景的基准约束问题的处理速度方面优于其他方法，如两个机械臂的逆运动学、路径规划、RoboCup 中的标准情况以及群体行为。在此认为这是由于所评估的问题具有相对简单的布尔结构，因此 SMT 求解器不能利用 SAT 求解部件的优势。在9.3节中将再次采用该假设。

局部搜索评估是基于 Hansen 和 Ostermeier[66] 提出的进化策略 CMA-ES（协方差矩阵自适应进化策略）以及 Riedmiller 和 Braun[136] 提出的弹性传播（Rprop）并结合自动微分进行的。在具体实现上，利用一种 Shtof[153] 提出的改进 AutoDiff 来实现自动微分。基于 Rprop 的搜索在性能上稍优于进化策略。

为了只利用数值局部搜索来求解 CSP，需要进行转换以产生一个需要优化的函数。为此，对输入公式进行转换以使得通过随后应用德摩根定律（de Morgan's LaW）和双重否定消除律，只能在不可分行为之前发生否定，之后通过转换 T 将

所得公式映射到一个函数。在参考文献［155］中所测试的四种转换函数中，下列转换函数的性能最佳。

定义 9.2 公式转换：

$$T(\phi \wedge \psi) = \Sigma_{\wedge}(T(\phi), T(\psi))$$
$$T(\phi \vee \psi) = \max(T(\phi), T(\psi))$$
$$T(a<b) = <*(a,b)$$
$$T(a>b) = <*(b,a)$$
$$T(\neg(a<b)) = \leqslant *(b,a)$$
$$T(\neg(a>b)) = \leqslant *(a,b)$$
$$T(\neg(a \leqslant b)) = <*(b,a)$$
$$T(\neg(a \geqslant b)) = <*(a,b)$$
$$T(a=b) = T(a \geqslant b-\varepsilon \wedge a \leqslant b+\varepsilon)$$
$$T(a \neq b) = T(a<b-\varepsilon \vee a>b+\varepsilon)$$

式中 $\Sigma_{\wedge}(a, b) = \begin{cases} 1 & a=1 \wedge b=1 \\ \min(0, a) + \min(0, b) & \text{其他;} \end{cases}$

$$\frac{\delta}{\delta x_i}\Sigma_{\wedge}(a,b) = \frac{\delta}{\delta x_i}(a+b)$$

$$<*(a,b) = \begin{cases} 1 & a-b<0 \\ b-a & \text{其他;} \end{cases}$$

$$\frac{\delta}{\delta x_i}<*(a,b) = \begin{cases} 0 & a-b<0 \\ \frac{\delta}{\delta x_i}(b-a) & \text{其他;} \end{cases}$$

$$\frac{\delta}{\delta x_i}\max(a,b) = \begin{cases} \frac{\delta}{\delta x_i}a & a \geqslant b \\ \frac{\delta}{\delta x_i}b & \text{其他;} \end{cases}$$

函数≤ * 的定义类似于< * 。

由此，结果为 1 类似于一个满足约束，而不可行的点对应于低于或等于 0 的值。由此产生的函数梯度中包括指向可行区域的不满足约束的相关信息。这种转换并结合梯度方法（如具有自动微分的 Rprop）可软实时地求解参考文献［155］中的各种约束问题。图 9.1 给出了所产生的函数。

在实际场景中，已知每个变量的边界。例如，在机器人足球中，位置变量不能随意远离比赛场地，并且角度范围为 $-\pi \sim \pi$。因此对于 CNLCSP 中的每一个变量，都可假设一个包含所有相关解的合理区间。这些边界可看作约束的注释或通过区间传播从约束中推导。表 9.1 给出了一个根据目前所介绍原理的基本求解

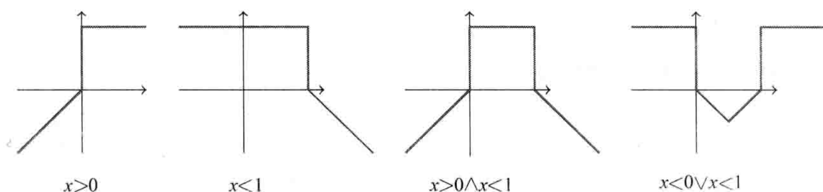


图 9.1　转换约束示例

器的伪代码。该求解器进行以不同随机点初始化的局部搜索，直到达到实验的上限（Maxtries）或找到一个解（$f(p)>0$）。表中还给出了相关搜索参数的默认值，包括与每个域大小相关的初始步长 s_{init}、梯度符号保持不变时步长增长因子 inc、梯度符号变化时步长减小因子 dec 以及最小步长 minV 和最大步长 maxV。

然而应用局部搜索法求解 CSP 的最大缺点是内在的不完备性。即使是一个确定性问题，利用局部搜索法也可能无解。更重要的是，局部搜索法不能降低约束问题的不满足性。因此，相应的求解器应与之前介绍的弱守护解释相结合。但由于这些求解器是用于软实时条件下，因此该问题可能不是特别重要。如果在一定时间段内无法求得解，所存在的解也不相关。在 9.4 节中，将详细讨论该问题。

表 9.1　基于 Rprop 的局部搜索

```
Solve(csp) {
  // parameters: s_init = 10^-3, inc=1.2, dec=0.5;
  //  minV = 10^-12, maxV = 10^10
  f := transform(csp);  // transform the problem into a function
  let p^min and p^max be the vectors of the lower (upper) bounds, respectively;
  d := number of dimensions of the csp;
  i := 0;
  while(++i < maxTries) {
     p := random point ∈ [p_1^min, p_1^max] × ... × [p_d^min, p_d^max];
     j := 0;
     let s and h be vectors with d elements;
     set all s_k := s_init · (p_k^max − p_k^min);
     set all h_k := 0;
     while(++j < maxIterations ) {
        if (f(p) > 0) return p;
        g := ∇f(p);
        for(k:=0; k<d; k++) {
           if (g_k · h_k > 0) s_k := s_k · inc;
           if (g_k · h_k < 0) s_k := s_k · dec;
           s_k := min(max(s_k, minV), maxV);
           p_k := p_k + sgn(g_k) · s_k;
           p_k := min(max(p_k, p_k^min), p_k^max);
        }
        h := g;
     }
  }
  return FAILURE;
}
```

9.3　重构 SMT 求解器

之前，本书声称在有关机器人场景的约束问题上，局部搜索法的性能要优于 SMT 求解器，既然如此，那么对于一个布尔结构足够复杂的问题，该方法应更加突出。在此，利用下列的问题发生器来对该假设进行评估，该发生器基于参考文献［155］中所述的正弦发生器，而该正弦发生器又是基于 Shang 等人[152]提出的发生器：

$$g(n,\ l,\ d) = (\phi,\ X,\ C)$$

式中　X——包含实数域的 d 个变量；

C——包含 l 个如下不等式：

$$k\sum_{i=1}^{3}\prod_{j=1}^{3}a_{ij}\sin(2\pi x_{ij}+c_{ij}) < \theta$$

式中　$k = \dfrac{1}{\sum_{i=1}^{3}\prod_{j=1}^{3}a_{ij}}$，

a_{ij}——在［-1，1］中均匀部分的随机值，

b_{ij}——在［-2π，2π］中均匀分布的随机值，

x_{ij}——X 中随机选择的变量，

θ——由随机采样测量的阈值，使得约束的可行区域近似为整个域的一半。这类似于由一个单一命题变量分解纯 SAT 问题的解空间的比例。

● ϕ 是一个包含 n 个子句的随机 3SAT 公式，其中每个命题变量 P 可唯一确认 C 中的一个约束。

● 以下列方式来保证 CSP 的可满足性：设 $\mathbb{R}^d$ 中一个随机点 $\vec{s}$ 作为 X 的价值函数 v。设 P' 为通过扩展 v 估值为 $\top$ 的命题变量集合，然后对于 ϕ 中每个子句，选择一个随机文本，若其命题变量位于 P'（不在 P' 中），则设为正（负）。

参考文献［155］中评估的 SMT 求解器并不适用于这种约束系统。AB 求解器[6]不能处理三角函数（这在机器人场景中非常重要），iSAT[48]对该基准约束问题的性能较差。在此推断这是由于解析器存在缺陷而不能正确识别重复发生的事件。但由于 iSAT 源代码不公开，这纯粹是猜测。

在此，对一个简单的 SMT 求解器进行评估，该求解器采用 Moskewicz 等人[102]提出的 DPLL 算法 Chaff 对命题变量 P 进行赋值。根据该赋值，构建一个纯粹的联合约束系统，并利用 9.2 节介绍的局部搜索法进行求解。局部搜索法只能重启一次。在其无法求得解的情况下，可调用 Chaff 算法来产生下一个命题模型，并不断重复该过程。在尝试过所有命题模型的情况下，两种求解器均复位，并重新开始执行该过程。在后面的章节中，将该求解器称为 Chaff + Rprop($\Sigma_\wedge$, max)，来表示所采用的算法和转换函数。

尽管利用这种 SAT 求解器和局部搜索法的组合方法使得局部搜索法可求解更加困难的约束问题，但由于联合约束问题是原始问题的一种特殊情况，该方法可以两种方式快速补偿。首先简化用于评估解空间中单个点的公式，其次局部搜索法不必处理会导致误导性场景的分裂，如参考文献［155］中所述。当然，在这种简单的方法中，两个求解器相对独立，相互之间几乎没有信息交换。一种更复杂的 SMT 求解器应能够交换更多信息，如冲突解释。

图 9.2 给出了 n 变化时由 $g(n, 50, 25)$ 产生的约束满足问题的求解结果。最大求解时间设为 30min。约束比是指子句和文本个数之比，在该情况下为 $n/50$。图中每个点都是经过 100 次实验后平均得到的。所有实验都是在 Linux 2.6.38 系统的单线程 Intel® Core i7 930 CPU（2.8GHz）计算机上进行的。

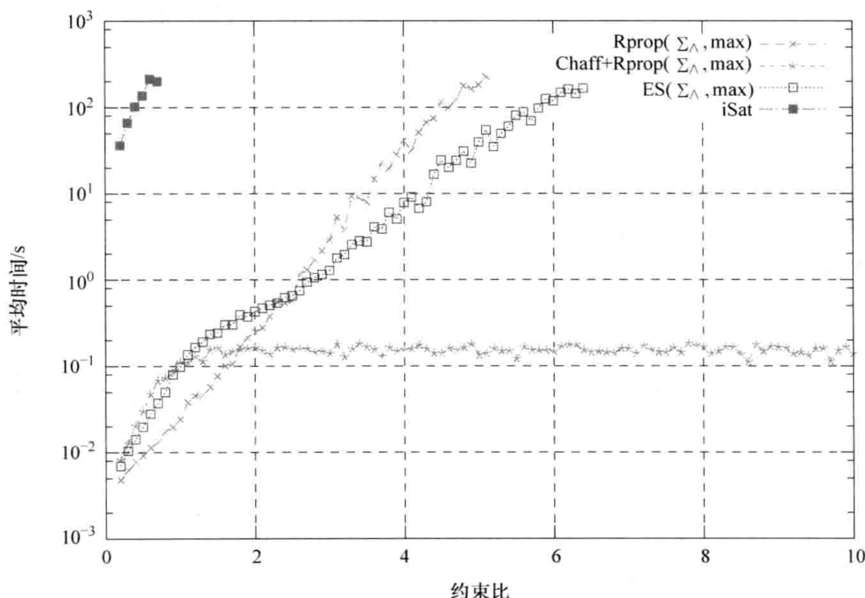


图 9.2　3SAT 正弦测试情况下局部搜索法和 SMT 求解器的性能

首先，基于 Rprop 搜索（Rprop($\Sigma_\wedge$, max)）和基于 CMA-ES 搜索（ES($\Sigma_\wedge$, max)）的性能都与参考文献［155］中的结果相同，尽管此处的问题更难。其次，尽管不同文本集为常数，局部搜索的性能还是要优于 SMT 求解器 iSAT[48]。当约束比小于 1 时，简单 SMT 求解器性能要比单纯的局部搜索法较差，但当约束比大于 2 时，其性能就优于局部搜索法。之后，对于不同的约束比，Chaff + Rprop($\Sigma_\wedge$, max）的性能基本不变。

由此可得出结论，与非线性约束的处理时间相比，SAT 求解时间并不显著（在

约束比大约为4.24时达到峰值[30,154])㊀。其次，约束比很低时，SMT方法对于局部搜索将比原始方法产生更难的问题。

当然，这是一个高度人为性的测试情况，主要是用于表明SMT的求解能力。而且如果可将某些性能成功地应用到实际问题中，本测试也展示了巨大潜力，然而这就是重点。图9.3给出本实验中每个求解器执行局部搜索的次数。Chaff + Rprop（$\Sigma_\wedge$, max）的运行次数在约束比为2时达到12.5，之后就保持不变。这意味着在求得解之前，平均只产生6~7个命题模型。假设在约束比为2~4时命题模型的总个数应更多，这就表明对于由$g(n, l, d)$产生的任何正弦文本组合，存在解的概率很高。

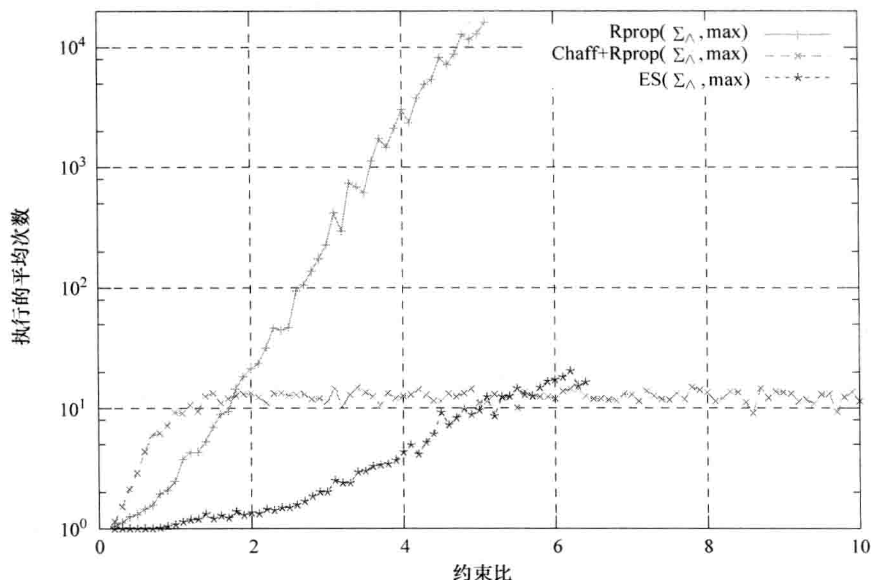


图9.3 3SAT正弦测试情况下执行启动（重启动）的次数

在更实际的场景中，可能并非如此，并且如果有多种形式的布尔结构，则这种简单的SMT方法很快就不可行了。下面的示例可勾勒出这种场景。

示例9.1 在机器人足球中，利用四个机器人来应对对方任意球的一个合适的策略可大致表述如下：

- 所有机器人均在场内。
- 所有机器人都位于对手罚球区域之外。
- 每个机器人都位于以足球为中心、半径为3m的圆外，或位于本方罚球区域之内。

㊀ 参见Russell和Norvig[139]的相关研究。

- 如果有某个机器人位于本方罚球区域内，则这是唯一的一个。
- 没有一个对手能被一个以上的机器人拦截。
- 当位于对手和足球之间时，每个机器人都必须要拦截四个对手中的一个，除非由于其他规则而无法实现。
- 一个机器人必须看着球，即机器人自定位以使得球与第二接近球的对手成一条直线。
- 机器人可在本方球门前组成一条防守线。
- 机器人必须观测相互之间的最小距离。

由于相应的约束系统非常繁琐，在此不具体描述，然而在合取范式中，该约束系统包括 90 多个不同的基本行为，并在大约 220 条子句中发生 2000 次以上。Rprop($\Sigma_\wedge$, max) 求解该问题需要大约 30ms，而 Chaff + Rprop($\Sigma_\wedge$, max) 运行 5min 后终止执行。

有效求解该约束问题的关键在于需要将信息从局部搜索后向传播到 SAT 求解器。这一思路是作为 DPLL(T) 方法由 Ganzinger 等人提出的[53]，其中对于可增量求解、回溯并对不一致进行解释的理论 T，Davis-Putnam-Logemann-Loveland 算法[36] 与求解器相结合。然而据目前所知，iSAT 是唯一的一种可处理含有超越函数的连续非线性约束的 SMT 求解器。在此将该问题作为今后的研究内容。下节将在介绍其他求解器特征（如解跟踪以及与其他机器人的求解器相配合）时考虑 SMT 求解技术。

9.4　实时性考虑

在实时场景下，获得一个约束查询结果的效用会随时间的推移而逐步变差。在某一时刻 $t+\Delta t_1$，时刻 t 时的查询结果就无用了。另外，在 $t+\Delta t_2$ 时刻，进行一个新查询，并且理想情况下，算法不再求解第一次查询。然而在许多情况下，由于所研究的问题几乎没有变化，因此新查询与之前的查询非常相似，而只是感知数据有一些微小变化，从而使得约束查询中的常量微小变化。图 9.4 给出了对连续查询解的效用所产生的影响。

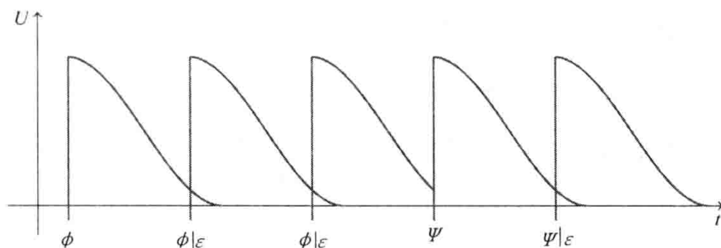


图 9.4　随时间变化的查询解的效用（查询三个 ε 等效的公式之后是一个新公式）

定义9.3 当且仅当 $\vec{x}=\vec{x}'$、$\phi=\phi'$且 C'可通过用 k'替换每个常数 k 使得 $|k-k'|\leqslant\varepsilon$ 后由 C 获得时，认为两个 CSP（ϕ，$\vec{x}$，C）和（ϕ'，$\vec{x}'$，C'）为 ε 等效。用 $\phi|_{\varepsilon}$表示与 $\phi\varepsilon$ 等效的所有公式，即 ϕ 的 ε 邻域。

示例9.2 在机器人足球领域，执行行为的频率通常为30Hz，因为这是机器人中最重要的传感器（摄像头）频率。这些行为每隔33ms进行一次查询。可能会导致相关约束满足问题显著变化的规划树几乎变化。在2009年的RoboCup中，平均每3.5s变化一次[157]。假设一个机器人自定位来接传球的任意球场景，在这种情况下，每次迭代都需要进行描述目标位置的约束查询。经过几秒后，裁判员示意比赛继续，然后就开始传球。由于机器人要尽快地得到球，因此只有这个时候约束问题才变化明显。一旦机器人接到球，约束问题再次变化，这次是描述能够进球得分的位置。

尽管约束系统是非线性的，但也可认为如果 $\vec{p}$ 为一个约束的解，则任何 ε 等效的约束问题都可在 $\vec{p}$ 的邻域找到一个解。在此，将利用这种关系的能力称为跟踪。

跟踪不仅可极大地减少求解查询的时间，而且还可使得解随时间变化稳定，这是一个非常重要的特性。由于查询的解通常用于获得执行命令（如驱动到一个约束位置），因此为平滑有效地执行，要求解应尽量稳定。即使在感知数据具有噪声的情况下，解也不会随时间变化而振荡。

动态约束满足问题，即 Brown 和 Miguel[16] 所深入讨论的随时间变化的约束问题，他们主要是考虑采用一种局部修补的概念，即在之前解的邻域内寻找一个解。另外他们还讨论了增加预测的可能性[172]，即根据所记录的信息进行搜索。然而上述方法是针对离散约束系统，而本书主要是考虑连续约束系统。近年来，Nguyen 和 Yao[110] 研究利用遗传算法来解决动态连续约束优化问题，他们提出了一种通过成员吸纳可行域之外的个体加入参考人群的可行域跟踪方法。该方法主要是针对遗传算法，而很难适用于其他方法。

很多求解器都能够通过时间来跟踪解，例如可以直接执行初始化一个起始点的局部搜索。第一个起始点就直接设为之前的解，然后理想情况下，局部搜索只需几步就能找到一个解。

一个 SMT 求解器（如 Chaff + Rprop($\Sigma_{\wedge}$，max)）还可以下列方式进行扩展来跟踪解：

- 缓存产生最后解的命题模型以及最后解。下一步，检查缓存解释是否也是一个模型。如果是的话，则用该模型以及缓存解来求解非线性问题。直观上，该方法的优点在于命题解释不随时间的变化而变化，因此约束满足问题所产生的行为可能更加协调一致。但这是比较困难的，在某些情况下，若约束问题的结构发生变化，则不可能从一个查询到下一个查询匹配文本。
- 另外，还可缓存解，并在第一次运行局部搜索时完全省略 SAT 求解步骤，

一旦搜索失败，则只能采用 SAT 求解。这虽然可避免公式结构变化的问题，但不能保证命题解释的稳定性。

● 最后，可缓存解，对于一个新查询，首先要根据该解来计算相应的一个解释，并利用该解释来初始化命题模型的局部搜索，因此就避免了从不同查询中识别文本的问题，然后所采用的局部搜索可使得获得一个模型所需的变化最小。Verfaillie 和 Schiex[173]提出了一种称为局部变化的算法。

这样，无论是否采用 SMT 求解器，都只需保存解来实现跟踪。一个约束满足问题的所有变量都可以唯一确定，这是因为这些变量是规划变量或对应于智能体函数流，因此通过缓存变量—值元组可以更好地求解，这样就允许即使在某些问题的维数发生变化时也能跟踪。对于查询中的每一个新元素（即变量），可对缓存的解附加一个随机值，这类似于根据缓存解并选择一个新的起始点都无法求得解的情况。

由于为使得变量尽可能相关，从规划树的任何部分都可进行查询，因此对于状态和规划类型的绑定，可在最高层上进行缓存。如果适用的话，8.7 节中介绍的遍历构成一个查询的规划树的函数已经在更高层上代替了变量，因此利用 CSP 的变量及其计算值可直接进行缓存。

相应地扩展表 9.1 中的算法也是很简单的。在首次运行局部搜索之前，不是以一个随机点来初始化搜索，则对于约束问题中的每个变量，都缓存查询一个值。如果没有值，即没有缓存，则采用一个随机值。求得解之后，相应的值保存在缓存中。9.5 节中，将讨论如何扩展缓存的概念，使得在多智能体领域中可提供一定程度的协调。

9.5　协作

在 pALICA 中，有关每个规划中分配的一致信念已足以实现协调一致的执行。但在约束满足问题中，已不再是这种情况。由于是多个智能体执行一个规划，因此每个智能体应独立地定期求解 CSP。因为单个智能体的行为取决于 CSP 的解，因此在整个团队中这些解应等价或至少相似，使得团队协调一致地执行程序。然而由于以下几种原因，会导致团队中的解可能存在偏差：

多解　典型的连续约束问题都具有多个解，经常甚至是有无穷多个解。如果对一个特定解没有给定优先级，则智能体可获得完全不同的解。为避免产生这种问题，可在约束满足问题中增加一个目标函数，从而产生一个约束优化问题。在 9.6 节中将详细讨论这一类约束问题。然而约束优化要比约束满足难得多，因此非常需要一种保证协调一致的且计算量较小的其他方法。

对称性　存在多解问题的一个特定原因是对称性。例如，如果假设机器人足球场景中的两个机器人 a 和 b 用于拦截对手 c 和 d，在没有任何限制条件下，a 的

结论可能是 a 拦截 c，而 b 拦截 d，但 b 的结论可能是 a 拦截 d，而 b 拦截 c。这是一个具体的任务分配问题，只是该问题集成到一个 CSP 中。同时，利用目标函数可破坏对称性，或者优先选择一种可实现协调一致且代价更小的破坏对称性方法。

感知噪声　由于感知信息具有噪声，每个机器人对环境的观察少许不同，从而导致团队中的解集合也不同。然而除非智能体的信念出现分歧，否则这些集合的交集为空，因此可假设存在一个团队一致认可的解。如果该解存在，则对于合作方法应选择这样一个解。

交叉 CSP　在 ALICA 中，每个智能体都各自具有一个取决于智能体所具有的规划以及所遍历的状态转移的约束库，而且智能体可以查询不同变量。因此尽管团队中的 CSP 查询集合是由某些变量连通的，但可能在维数和子句上有很大不同。若相应的可行域交叉比可行域联合要小得多，则很难实现协调一致。显然，如果交叉为空，则无法协调一致。

在讨论上述情况的可能解之前，设定一些约束满足问题之间相互关系的术语。

定义 9.4　给定两个 CSP $c_1=(\phi, X, C)$ 和 $c_2=(\phi', X', C')$，认为：

- 当且仅当 $X\cap X'=\Phi$ 时，c_1 和 c_2 相互独立。
- 当且仅当对于 c_1 的解 η 和 c_2 的解 η'，使得 $(X\cap X')\eta=(X\cap X')\eta'$ 时，c_1 和 c_2 相互兼容。
- 当且仅当对于 c_2 的所有解，存在 c_1 的一个解 η 使得 $(X\cap X')\eta=(X\cap X')\eta'$ 时，c_1 包含 c_2，记为 $c_1>c_2$。
- 当且仅当 $c_1>c_2$ 且 $c_2>c_1$ 时，c_1 和 c_2 相等。

一般而言，所有独立的 CSP 都是等效的，并且所有独立可行的 CSP 是相互兼容的。而且任何 CSP 都包含不可行的 CSP，如果另一个 CSP 包含一个可行的 CSP，则两者兼容。

设 $C=\{(\phi_1, X_1, C_1), \cdots, (\phi_n, X_n, C_n)\}$ 是智能体团队考虑的一组 CSP。处理该集合的一个简单方法以及所引发的问题都是构建交集 $\Phi=(\wedge\phi_i, \cup X_i, \cup C_i)$，对其集中求解并分配解，显然每个独立的 CSP 都包含交集。然而这会产生一个瓶颈，至少暂时是一个单点失败。而且在动态环境中，可能导致在获得新的感知信息之前无法与单个 CSP 及时通信，并且全局 CSP 在求解之前已失效。

或者，类似于任务分配中的 pALICA 方法，每个智能体可定期广播解，然后将接收到的解作为变量—值元组集成到缓存中，因此缓存中就累积了所有机器人计算的解。根据 CSP 和信念基，这些解可紧密地聚类或分散在解空间。由于情况动态变化，一个智能体不能盲目地信任一个接收解，而是在使用该解之前先检查是否与当前的约束库相矛盾，然而假设 CSP 在团队中没有太大变化，则解可能相近。这意味着可利用所接收的值从求解的局部搜索中推导初始点。

在这些解相差较小的情况下，以每个接收的解作为一个起始点会导致性能降

低，这是因为可能多次访问解空间的同一区域。另一方面，在这些解分布在整个问题场景的情况下，将所有缓存值合并为一个起始点势必会过度泛化，因此这些解应利用动态个数的聚类来积聚。当解比较分散时应采用许多初始点，因而团队中的冲突程度较大，从而使得每个智能体评估团队的许多解或全部解，并进行局部比较。在这些解已收敛到较小区域时，所需的起始点很少。

实现该行为的聚类算法有很多，如基于 Ward[176] 研究工作的层次聚类法，或 Schubert 和 Sidenbladh[146] 提出的结合贝叶斯滤波的有序聚类。在该情况下，将所得的聚类中心作为局部搜索的起始点，因此与算法的时间效率相比，聚类结果的质量不太重要。而且对于所有参与的智能体，解应该不同。因此在此采用一种简单的基于距离的聚类方法，可产生大量聚类中心。然而该效应只会导致多次搜索某个区域，而不应显著降低整体性能。

假设缓存中存储了每个智能体和每个变量的最后接收值，以及本地智能体的最后计算值。表 9.2 给出了一种简单但快速的聚类算法伪代码。采用如下的符号约定：C 表示缓存，$C(a, x)$ 表示智能体 a 计算的变量 x 的缓存值。最后，$C(a,x) = \perp$ 表示在缓存中不存在解，即 a 没有接收到 x 的任何值。

表 9.2　缓存解的聚类

```
InitialPoints (v⃗) {
//v⃗ is the ordered list of queried variables
Let r⃗ denote the size the domains of v⃗.
Let θ be a fixed threshold, i.e., 10^-3.
R := Φ;

Let M be a list containing for all agents a ∈ 𝒜:
    c⃗ := (C(a,v_1),...,C(a,v_dim(v⃗)));
Let M be uniquely ordered according to an ordering over 𝒜.

remove all (⊥,...,⊥) from M;

while(M ≠ Φ) {
    s⃗ := (⊥,...,⊥);
    k⃗ := (0,...,0);
    foreach(c⃗ ∈ M) {
        d := 0;
        t := 0;
        foreach(i ∈ [1,dim(v⃗)]) {
            if(c_i ≠ ⊥ ∧ s_i ≠ ⊥) {
                d := d + ((c_i - s_i)/r_i)^2;
                t := t + 1;
            }
        }
        if (t = 0 ∨ d/t < θ) {
            foreach(i ∈ [1,dim(v⃗)]) {
                if (c_i ≠ ⊥) {
                    if (s_i = ⊥) {
                        k_i := 1;
```

（续）

```
                    s_i := c_i;
                } else {
                    s_i := (k_i s_i + c_i)/(k_i + 1);
                    k_i := k_i + 1;
                }
            }
        }
        remove c⃗ from M;
    }
  }
  if ((∃s_i ∈ s⃗_i) s_i ≠ ⊥) R := R ∪ {(Σk_i, s⃗)};
}
Sort R descending by first tuple element;
return R;
}
```

尽管所给出的算法在准确率上无法与现有的聚类方法相比，但该算法计算速度快，且足以使得聚类点在解空间中相互接近，而且还可处理部分解，即不包括所有查询变量的CSP的解。当两个智能体求解交叉CSP，或一个智能体通过附加变量对CSP扩展时，可出现这种情况。该算法不能对所处理点的顺序进行交换，因此解矢量列表 M 的排列顺序是唯一的，从而使得每个智能体以相同顺序来处理解。

运算结果是 $(\mathbb{R}\cup\{\bot\})^n$ 中的一组有序矢量集，其中 $\bot$ 表示不能从缓存中得到没有合适的值。为获得局部搜索的初始点，这些值应由随机数代替。对该集合排序以使得个数较多的聚类在个数较少的聚类之前进行。在一个行为中的结果也类似于多数表决的方法，即使得每个智能体优先选择许多其他智能体计算的解。在这种聚类的基础上，可将表9.1中的局部搜索进行扩展，产生表9.3中的算法。

该算法从缓存中得到聚类结果，并根据这些结果进行迭代，直到在局部搜索中求得一个解。如果执行完所有的初始点，将继续以随机点执行。一旦达到设定时间就终止该过程，这时的解不相关，根据9.4节中的实时性考虑，CSP认为这些解不可行。然而如果尚未求得解，则该算法应对第一个缓存值运行一次局部搜索，并根据随机点也至少运行一次局部搜索。这样就可保证能够跟踪最相关的解（即第一个解），并且在每个执行周期中至少进行一些尝试，直到求得一个解。因此为搜索整个解空间，该算法可避免由超时定义的软实时约束。

由于是对所有聚类解进行迭代，因此可根据智能体的总体排序来确定解的优先级。该排序决定了智能体的优先级，并原则上可根据感知信息的质量与相关性来动态计算，然而最重要的是整个团队应已知该排序。在此，并不讨论如何实现这种动态排序，而是假设一种简单的静态排序（如根据唯一的标识）。

该算法可解决10.3节中所述的对称解、多解以及抗噪问题。而且还可利用团队的分布式计算功能来求解，从而有效地解决了无法在一个周期内分布式求解

CSP 难题的问题。这种情况下，智能体利用随机起始点搜索整个空间，直到一个智能体找到一个可行解，然后分布该可行解。

表 9.3 通过缓存进行协调的局部搜索

```
Solve(csp) {
  // parameters: s_init = 10^-3, inc = 1.2, dec = 0.5
  //  minV = 10^-12, maxV = 10^10
  begin := now;
  f := transform(csp);  // transform the problem into a function
  let p^min and p^max be the vectors of the lower (upper) bounds, respectively;
  d := number of dimensions of the csp;
  i := 1;
  M := InitialPoints (vars(csp));
  while(true) {
     p := (⊥, ..., ⊥);
     if (i ≤ |M|) {
         p := M_i;
     }
     foreach(p_i ∈ p) {
         if (p_i = ⊥)
             p_i := random value ∈ [p_i^min, p_i^max];
     }
     j := 0;
     let s and h be vectors with d elements;
     set all s_k := s_init · (p_k^max − p_k^min);
     set all h_k := 0;
     while(++j < maxIterations) {
         if (f(p) > 0) {
             insert p into cache;
             return p;
         }
         g := ∇f(p);
         for(k=0; k<d; k++) {
             if (g_k · h_k > 0) s_k := s_k · inc;
             if (g_k · h_k < 0) s_k := s_k · dec;
             s_k := min(max(s_k, minV), maxV);
             p_k := p_k + sgn(g_k) · s_k;
             p_k := min(max(p_k, p_k^min), p_k^max);
         }
         h := g;
     }
     if (time since begin > timeout) {
         if (i = 1) i := |M|+1; //do at least one free exploration
         else return FAILURE;
     }
     i := i+1;
  }
}
```

但是该算法不能解决交叉 CSP 的问题，尽管该算法可处理具有不同但交叉的变量集的 CSP 问题，但在可行域不同时，并不能保证团队收敛。这是一个不能轻易解决（否则该 CSP 问题就易于处理）的难题。也就是说，如果存在一个

解决交叉 CSP 问题的方法，这比通过构建交叉并局部求解要更加有效，也比本身是一个高效求解器的方法更有效。

显而易见，在给定通信延迟、非线性且求解器不完备性的条件下，没有一种不牺牲反应能力的更好方法。由于不完备性，求解器不能推导一个 CSP 的不可行性，基本上只能测试一个给定赋值是否构成一个解。另外在未知 CSP 且没有局部测试点或查询其他机器人的情况下，一个智能体无法知道一个解释是否是另一个智能体认为有效的 CSP 的一个解。由于延迟而导致查询需要花费大量时间，同时也会造成问题发生变化的可能。另一方面，如果查询智能体已知 CSP，这样就可解决两个 CSP 的交叉问题。这比分别考虑两个 CSP 更有效，否则就应通过问题分解来构建一个更好的求解器。值得注意的是，如果假设相对于环境的动态变化，通信延迟较小，则这些消息可用于实现 Petcu[121]所提出的分布式求解与优化技术。

针对这种类型的协调合作问题，另一种方法是通过监视缓存来检测由交叉 CSP 产生的一个持续冲突，并且一旦发生冲突，就将决策协议切换到另一种以牺牲一定程度的反应能力来解决冲突的协议。例如，可指定一个智能体来构建交叉，求解后并广播解，这类似于第 6 章中介绍的局部领导者协议。或者，可假设每个智能体对部分 CSP 进行显式控制，并在团队中交换解，这类似于参考文献 [121] 中的 DCOP 方法。在此，将这种处理交叉 CSP 的方法作为今后的研究工作。

9.6 约束优化

在前面的章节中，主要考虑了约束满足问题，其中智能体行为取决于这些 CSP 的解集合。然而在许多场景中，可能不仅要描述一组解，还要根据某种准则选择其中的最优解。尽管这仍可以看作一个允许量化的约束满足问题，但作为约束优化问题的一个公式往往更容易求解。

布尔域的量化 CSP（QCSP）是 PSPACE 完备的[31]。据我们所知，目前还没有一种算法能有效处理连续域的 QCSP，以用于动态域中运行的智能体团队，尤其是存在非线性函数和超越函数时。目前，针对连续域 QCSP 求解器的研究工作主要是由 Goldsztejn 等人[60,59]进行研究。

在此，并不按照 QCSP 的思路，而是设定一个优化准则并结合 CSP 来产生一个约束优化问题（COP）。

定义 9.5 如果 $P=(\vec{x}, \phi, C)$ 为一个 CSP，f 为 $\vec{x}$ 到 $\mathbb{R}$ 域的一个函数映射成员，则 $P'=(P, f)$ 是一个 COP。P 的任一解 s 也是 P' 的解。P' 的一个最优解也是 P 的一个解，使得所有 P 的解 s' 存在 $f(s) \geqslant f(s')$。f 称为目标函数。

在此，只是针对单个目标函数的情况，因此该定义不包括多目标的优化。

扩展表9.1的局部搜索方法来处理优化问题是非常简单的。本质上，该算法在确定一个可行域之后并不是立刻终止，而是跟踪可行域内目标函数的梯度。缺点是对于一个单纯的约束满足问题，不能产生一个合适的终止条件准则。实际上，如果没有限定目标函数位于搜索空间，则由于自身的不完备性，该算法不能保证一定可求得一个全局最优解。

然而对于许多约束问题，目标函数的上限是事先已知的，或通过区间传播可计算得到。另外在许多场景下并不需要得到一个最优解，而只需得到一个足够接近的解就可接受，尤其是在给定严格实时性约束的条件下。因此假设对于任意目标函数f，都存在一个表示充分性阈值的f_{max}，使得对于目标函数为f的任意COP，一个使得$f(x) > f_{max}$的解x表示这是一个足够接近最优解的解，这时可终止搜索。当然f_{max}也可能等于∞，这时只有达到超时才会终止搜索。

示例9.3　对于一个约束n个机器人目标位置的CSP，一个合适的目标函数应使得机器人当前位置与其目标位置的平方距离和：$f(\vec{x}) = -\sum_{i=1}^{n}\mathrm{Dist}(\mathrm{Goal}(i), Pos(i))^2$最小。

这样就可使得机器人需要运行的距离最短，从而更快地达到目标状态。由于任意两点之间的最短距离为0，则$f(\vec{x})$的全局最优也为0，尽管机器人已达到可行位置，但只有该点才有效。另外由于感知噪声以及执行机构的准确性限制，机器人不能非常精确地自定位，因此应根据域选择一个限制范围，如$f_{max} = -0.1n$。

由于相对于目标值，不同初始点获得的解应能够相互比较，因此最优化还需要交换解或缓存解的一个稍微复杂的处理。交换解允许团队分布式地搜索整个空间，与单个智能体相比，这能够更快地获得最优解或近似最优解。

这样，一个智能体就应选择一个与最佳目标函数值相适应的解。然而为使得该解不随时间变化且在团队中稳定，应在一定程度上优先选择较早计算的解或其他多个智能体计算的解。在约束满足问题中，这可通过聚类解的优先级排序来实现。在此引入两个最优准则，即给定的目标函数，以及随时间变化和在团队中的一致性。由于域的动态性，这两个准则可能存在冲突。

可采用下列方式来处理冲突问题：

- 首先，优先选择最佳目标函数值，从而允许高度反应性的行为。
- 其次，以一个不同参数集（即最小步长更小，迭代次数更多）来执行首次运行的由表决个数最多的聚类中心初始化的局部求解器，这可增大跟踪一个最优解的概率。
- 第三，首次运行求解器得到的解具有较高的优先级，从而使得其他解必须至少在一定程度上优于首次运行的解。该种假设可有效抑制振荡。
- 最后，在存在一个目标函数的条件下，一旦优化了一个非约束问题，则

所得到的解释可作为约束问题的初始点，因此该算法只搜索目标函数最优解附近的可行域。

这些措施可有效避免求解一个大时变多目标优化问题的并发性，并允许在目标场景变化的情况下快速反应。在 10.3 节将对该技术进行评估。

表 9.4 给出了完整算法，内部的优化循环见表 9.5。值得注意的是，其中精确搜索以及不太精确搜索的参数都是临时选择的示例值。非约束问题的单次优化运行值得商榷，这是因为这可改善或和减损（参见参考文献［180］中关于该问题的讨论）。然而这可保证在每次迭代中，时间约束允许条件下可搜索目标函数值较高的区域。在此，认为这对整个约束问题的求解是有利的。

表 9.4 最优化和协调的局部搜索

```
Solve( satisfaction problem csp, objective function o, objective threshold o_t) {
 // parameters: significance threshold s_t = 10^-22
 begin := now;
 f := transform(csp); // transform the problem into a function
 let p^min and p^max be the vectors of the lower (upper) bounds, respectively;
 d := number of dimensions of the csp;
 i := 1;
 M := InitialPoints (vars(csp));
 p_best := FAILURE;
 while( true ) {
    p := (⊥,...,⊥);
    if (i ≤ |M|) {
        p := M_i;
    }
    foreach (p_i ∈ p) {
        if (p_i = ⊥)
            p_i := random value ∈ [p_i^min, p_i^max];
    }
    if (time since begin < timeout ∧ i = |M| + 1) { //do one unconstrained optimisation
        p := Rprop(p,1,o,⊥);
    }
    p := Rprop(p,f,o,i=1);
    if (p ≠ FAILURE ∧ (p_best = FAILURE ∨ o(p) > o(p_best) + s_t) {
        p_best := p;
        if (o is constant ∨o(p_best) > o_t) {
            insert p_best into cache;
            return p_best;
        }
    }
    if (time since begin > timeout) {
        if (i < |M| + 1) i := |M|+1; //do at least one free exploration
        else {
            insert p_best into cache;
            return p_best;
        }
    }
    i := i+1;
 }
}
```

表 9.5　单次最优化运行

```
Rprop(p,c,o, precise ) {
 //arguments: p – the initial point, c – the transformed csp, o – the objective function,
 // precise – indicating which parameter set should be used
 // parameters: inc=1.2, dec=0.5, s_init = 10^-3, maxV = 10^10
 if ( precise ) {
    maxIterations := 110, minV := 10^-15
 } else {
    maxIterations := 60, minV := 10^-11
 }
 let p^min and p^max be the vectors of the lower (upper) bounds, respectively;
 d := number of dimensions of the csp;
 let s and h be vectors with d elements;
 set all s_k := s_init · (p_k^max – p_k^min);
 set all h_k := 0;
 p_best := p;
 i := 0;
 while(++i < maxIterations ) {
    if (c(p) ≤ 0) {
        g := ∇c(p);
    }
    else {
        if (o is constant) return p;
        g := ∇o(p);
        if (c(p_best) ≤ 0 ∨ o(p) > o(p_best))
            p_best = p;
    }
    for (k:=0; k<d; k++) {
        if (g_k · h_k > 0) s_k := s_k · inc;
        if (g_k · h_k < 0) s_k := s_k · dec;
        s_k := min(max(s_k, minV), maxV);
        p_k := p_k + sgn(g_k) · s_k;
        p_k := min(max(p_k, p_k^min), p_k^max);
    }
    h := g;
    }
 }
 if (c(p_best) ≤ 0) return FAILURE;
 return p_best;
}
```

尽管对基于 SMT 的方法进行扩展以解决优化问题已超出本书的范畴，但在这里还是要讨论一下处理优化问题的可能性。首先，一个 SMT 求解器可在所有命题模型中迭代，并比较所得的解。随着模型个数的增加，该求解器很快就不可行。其次，一个非约束目标函数的最优解可用于计算作为在命题邻域内迭代进行局部搜索的起始点的一个命题分配。最后，只要求得一个解，即可利用分支定界方法[32]通过迭代确定附加约束来缩小命题搜索空间。

9.7　约束与任务分配

ALICA 中 CSP 和 COP 均允许以一种简洁的方式来制定功能强大的一阶决策，而且还可有效求解该决策。然而在规划执行期间，智能体面临两个独立而交错的问题，即找到查询变量的一个赋值，并找到一个智能体与任务相关联的任务分配。这些解对智能体的行为有着主要影响。通常 CSP 是指分配给特定任务的智能体的动态特性，如在环境中的位置、续航能力或某些关节的当前配置。因此，任务分配会影响一个 CSP 的可行域，并对一个 COP 的最优解有影响。这些关系是由展开步骤（见定义 8.8）决定的。

理想情况下，这两个问题可统一。所产生的问题描述将使得取决于每个智能体的任务以及一组连续变量的目标函数最大化，从而满足一个布尔组合的约束。

处理这两个问题的一种简单方法是：在所有基数允许的任务分配中进行迭代，在所有满足 CSP 的任务分配中，选择效用最高的，然而该方法由于运算量过大而很快不可行。另外如果利用现有方法在首次迭代中没有求得全局最优解，则可在降低团队性能的基础上，从该方法中选择一种计算速度较快的任务重分配。

原则上，该问题可看作一个混合整数的非线性约束规划问题。近年来 Nema 等人[107]不断改进处理该问题的技术。然而确定一种适用于该问题的合适求解器仍是今后的主要研究内容，要求该求解器能够在严格时间约束条件下有效执行（如求解标准情况下的约束问题不超过 30ms）。Nema[108]的经验结果表明这些求解器并不适用于这种软实时场景。

可以交错执行任务分配的 A^* 搜索和约束求解，以使得搜索树的大部分都可能被提前裁剪。但从另一个角度来看，这两个核心问题的统一存在一个主要缺点。5.12 节中介绍的任务分配算法是合理且完备的。该算法与一种如求解非线性约束满足问题的局部搜索的不完备算法相结合，可导致一种不完备的任务分配，这是非常不希望出现的。

只要最有效的任务分配产生一个可行的 COP，使得其全局最优解至少与其他有效任务分配产生的所有 COP 的全局最优解同样好，则任务分配和约束优化的不相交处理可产生一种最优解。目前，这种特性只能通过仔细设定运行时条件和效用函数来实现。

9.8　本章小结

本章介绍了一种可随时求解很多类约束问题的求解器，即连续域中具有有理

函数和超越函数的非线性约束的布尔组合。我们认为该类问题适用于多机器人领域发生的各种不同问题。基于结合局部搜索和自动微分的求解器可提供线性时间内每个点的梯度。阐述了如何利用缓存跟踪时变解，并通过集成一种聚类算法将该缓存方法扩展为一种协调方法，对所有可能解进行多数表决。之后讨论了约束优化问题的相关性，并随后对算法进行扩展来处理该问题类。在第 7 章中所讨论的参考实现集成了该求解器，而且还在基于动机基准测试的场景中讨论了 SMT 求解技术的潜力。最后从统一的角度出发，确定任务分配和 CSP 求解为两个相关问题。

第 4 部分　评估

第 10 章　评　　价

在前面的章节中，介绍了一种多机器人编程语言 ALICA。首先在第 2 部分中从一个基本的命题式变量开始，接着在第 3 部分，对此在连续值约束下进行扩展。在此重新讨论第 1 章中所述的设计目标，并给出相应的评估结果。

首先，在 10.1 节中，介绍了 Carpe Noctem 团队在利用之前所介绍的方法进行团队行为建模时所积累的经验。根据这些经验所产生的某些设计模式已被证明具有较好的鲁棒性和可重用性。10.2 节主要讨论了非可靠网络条件下的鲁棒性，并给出了在机器人足球领域中具有不同程度的丢包和包延迟情况下的仿真结果。同时，还表明了在这些条件下进行冲突检验和冲突消解的效果。在 10.3 节中，对约束求解和优化技术的鲁棒性进行了评估。主要评估了随时间变化的噪声以及智能体团队在不同程度的感知噪声下求解约束问题的协调一致性。10.4 节描述了如何在外太空搜救的场景中应用 ALICA。此外，还介绍了一种多机器人协作探索某一区域的动态编队方法。最后，在 10.5 节，将 ALICA 应用于一个搜救领域的场景。主要是利用 ALICA 来协调和控制一个消防队，以应对城市中的多处灭火任务。将 ALICA 与一组任务分配问题的处理方法进行性能对比，并对 ALICA 在所参与的智能体个数方面的可扩展性进行了评估。

10.1　机器人足球建模

自 2009 年，Kassel 大学的 Carpe Noctem 机器人足球团队已成功应用了 ALICA[㊀]。这使得开发人员在应用 ALICA 实现规划和策略方面积累了宝贵经验。这些经验最终应用于具体的设计模式，证明在存在噪声、非可靠性通信以及机器人损坏等情况的某些领域中具有高效、可重用性和鲁棒性等特点。同时也阐述了 ALICA 的表现性。接下来，由于特定域描述并非本书内容的一部分，且更加正式的表示方

㊀ http://das-lab.net。

法并不能对示例有多大帮助，因此在此用一种伪正规方式来描述特定域公式。

10.1.1 强同步与弱同步

如5.13节所述，ALICA可通过采用显式语言元素来实现强同步，或利用机器人的信念条件来实现弱同步。图10.1给出了采用一个同步元素来实现强同步的情况，使得智能体只能分别从状态 z_0 到状态 z_2 以及从状态 z_1 到状态 z_3 来共同运动。若智能体之间的共同信念是必须满足 ϕ_1 和 ϕ_2，则智能体只能按照上述状态转移进行运动。

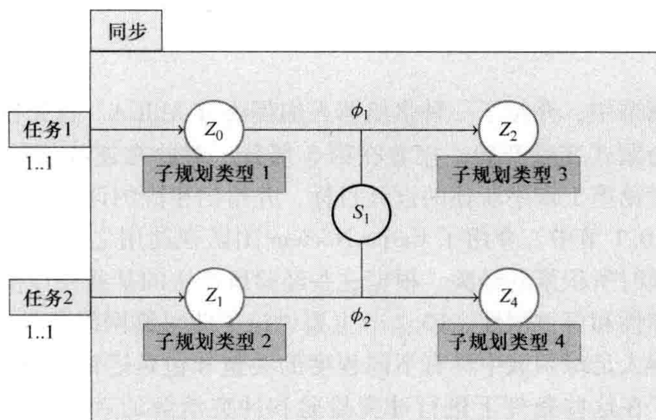


图10.1 ALICA中的强同步

由于通信开销过大，因此在RoboCup中很少采用强同步。与此同时，却经常采用弱同步（如描述一个传球时）。图10.2描述了一个类似的模式。在此，执行任务1的智能体首先根据状态转移运动，而在该智能体达到状态 z_2 之前，执行任务2的智能体一直保持等待。这种协同工作只需一条消息。另一方面，由于第二个机器人始终是在第一个机器人运动完成后才运动的，因此并没有实现真正的同步状态转移。这在许多场合中都是可接受的。同时，弱同步会更容易出错，例如在第二个智能体没有接收到来自于第一个智能体的规划树消息的情况下。ALICA中所采用的定期广播可在某种程度上缓解该问题的发生。

10.1.2 有限状态机与动态任务分配

ALICA建模语义的核心是三个相互对立的概念，即有限状态机、动态任务分配和约束满足与优化问题。从建模角度来看，任意组合这三个概念的能力是ALICA的一个优势。图10.3描述了仅利用动态重分配来制定的一个RoboCup策略。相应的效用函数 $\mathcal{U}$(121play) 可使得进攻队员与球之间的距离以及防守队员与己方球门之间的距离最小化，因此执行防守任务的优先级要高于执行助攻任务。相比之下，图10.4中所描述的进攻规划是一个纯粹的有限状态机。在给定

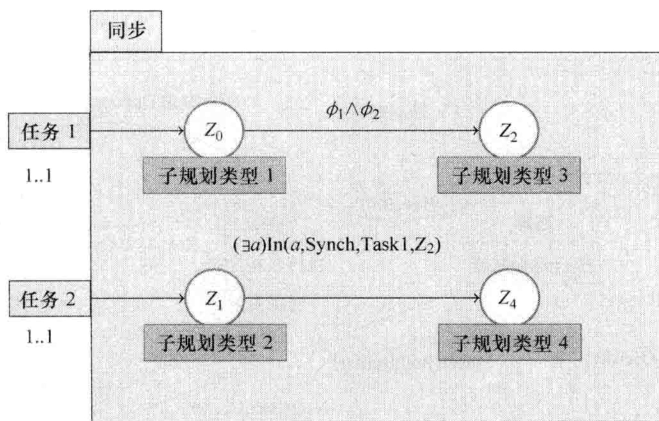


图 10.2　ALICA 中的弱同步

球和对手感知位置的条件下，该规划控制着机器人的动作。

上述两个示例表明了如何将有限状态机和动态任务分配这两个概念在同一个 ALICA 程序中成功地结合应用。同时这两个概念甚至还可以在同一个规划中结合应用，如图 10.5 所示。图中，两个机器人位于控球机器人的附近，以防止对方防守队员接近。根据场上信息，任务分配算法动态地对智能体分配这三个任务，而状态转移则使得保护机器人在跟随状态和拦截状态之间切换。

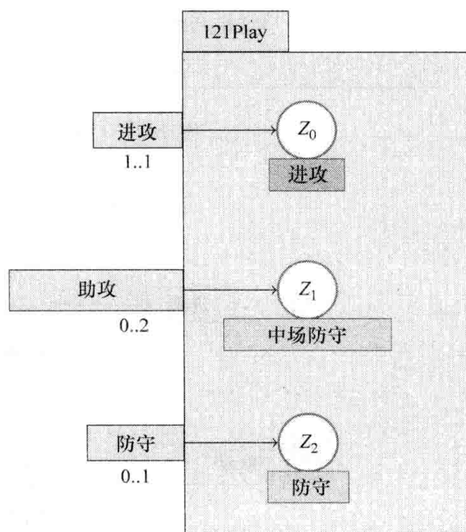


图 10.3　RoboCup 中的 1-2-1 策略

10.1.3　选择与执行

有限状态机、动态分配和决策相结合可产生另一种设计模式，通常称之为“选择和执行”。在此，团队执行一个规划类型并动态切换执行规划，直到满足某一条件，然后团队根据当时执行的规划类型转移到一个所选的状态，从而在条件为真时进行动态决策。在图 10.6 中利用在机器人足球中掷界外球的一个 ALICA 规范来描述上述情况。

在掷界外球规划中，对一个机器人分配守门员任务，而其他机器人执行 FieldPlay 任务。守门员独立地防守球门，其余场上队员既可以在状态 z_1 下朝各自

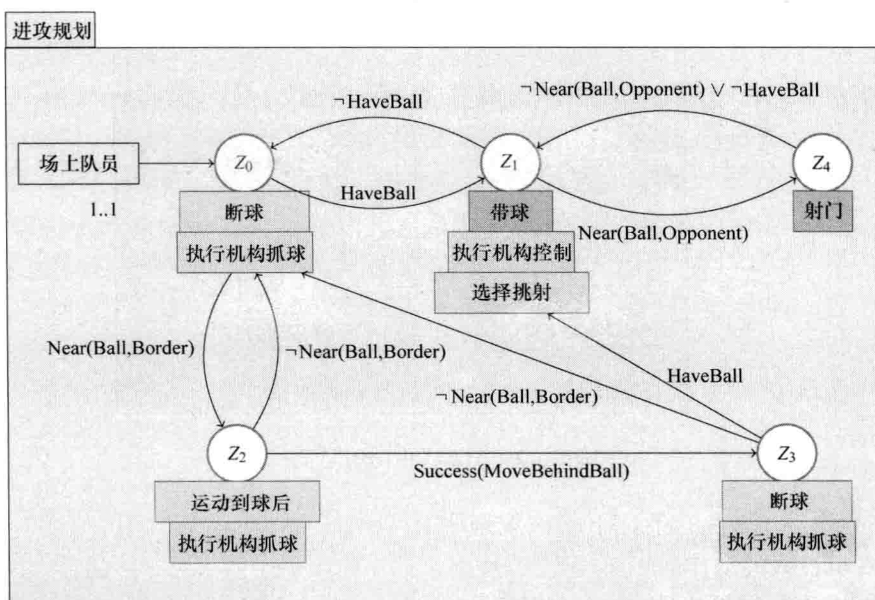


图 10.4　进攻机器人的 RoboCup 规划

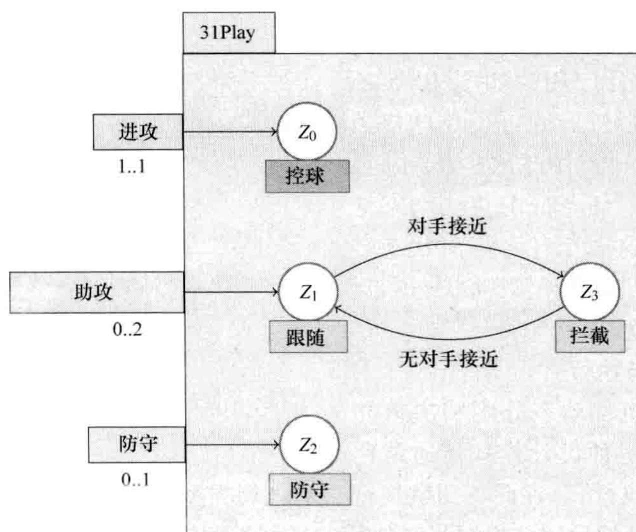


图 10.5　RoboCup 中的 3-1 策略

的位置运动，也可以在状态 z_2 下搜索位置未知的足球。在这种简单的示例情况下，掷界外球位置规划类型中包含四种不同的规划，每种规划都代表着一种不同的情况。团队根据足球位置、对手行为以及当前比赛得分情况来选择其中的一种

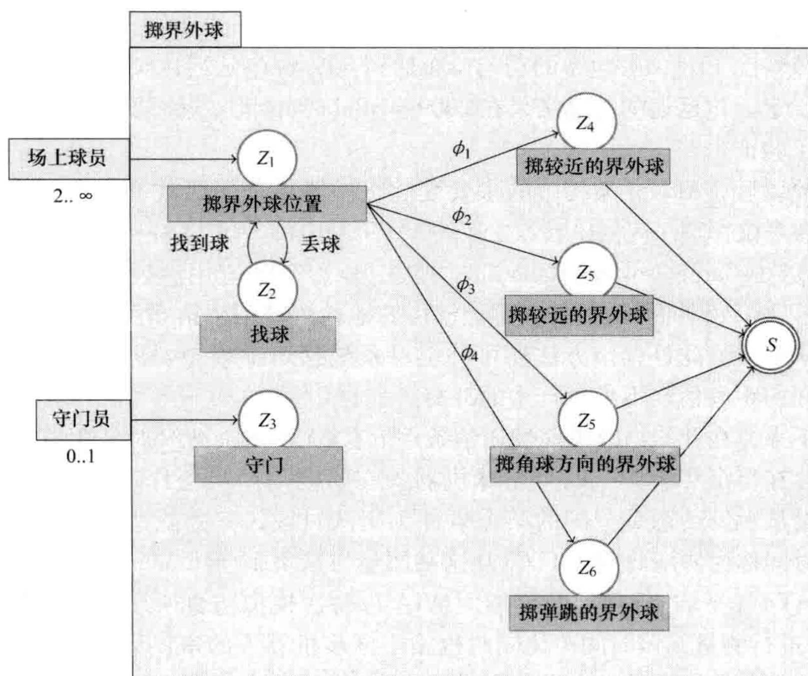


图 10.6 RoboCup 中掷界外球的规划

规划。只要比赛处于这种定位阶段，团队便可在这些规划中动态地切换。条件 $\phi_1 \sim \phi_4$ 均具有以下形式：

$$\text{Situation}(\text{Start}) \wedge ((\forall a, x, y)\ \text{In}(a, \text{ThrowIn}, \text{FieldPlay}, z_1) \leftrightarrow \text{In}(a, p_i, x, y))$$

式中 p_i——掷界外球位置规划类型中的一个具体规划。

这样一旦接收到裁判员的比赛开始信号，同时团队成员之间达成共识，则团队就会转移到由当前执行规划决定的下一个状态。如果已知产生了一个冲突，则首先解决该冲突，然后场上队员再进入执行阶段。值得注意的是，由于条件是在机器人发生状态转移时变化，因此这种结构并不与局部性原理相冲突。前置条件中的一个类似结构将会违背规划的局部性原理。

本节所给出的示例表明了在 ALICA 中建模的方便性及其出色的表现性。凭借 RoboCup 机器人团队所积累的经验，将逐渐出现一些类似于编程语言中模式的设计模式。

10.2 非可靠性通信

在参考文献［157］中首次给出了在没有冲突检验和冲突消解的情况下对

pALICA 的评估结果，着重关注在网络条件较差情况下的性能。结果表明在网络条件较差情况下（如丢包率的均匀分布达到50%，包延迟达到600ms），性能仍然表现稳定。包延迟的仿真结果在2009 年 RoboCup 世界冠军赛的真实环境条件下得到了验证[157]。

自此以后，ALICA 架构中的主要变化之一是增加了显式冲突检验和冲突消解。在参考文献［158］中首次介绍了 ALICA 中的冲突检验和冲突消解，并于2011 年初在 Carpe Noctem RoboCup 团队中开始得到应用。表明由于静态误差（即配置不一致）而导致的冲突，能够可靠地检测并利用第6 章所介绍的冲突消解机制来解决。此外，该方法还可补偿由系统感知误差（如机器人对与被感知物体之间的距离估计不足）产生的冲突。

接下来，在此给出非可靠性通信条件下有/无冲突消解时对 ALICA 性能的评估结果。在每个实验中，四个足球机器人在不同的网络条件下模拟运行 5min，然后测量从触发动态重分配的某个事件到团队对该任务分配达成一致所需的时间。该时间称为协调时间（TTC）。这种测量方法类似于 Kaminka 和 Tambe[82] 所提出的 ATA（平均协商时间）。然而 ATA 以每次模拟仿真所用的抽象时间单位“ticks”进行测量，该时间单位同时也用于同步机器人的审议周期。TTC 以秒为单位。此外，在仿真过程中没有人为地同步机器人的行为或感知信息。

除此之外，还测量了团队中不同信念状态的平均数，称之为信念计数值（BC），其范围为 1（完全同意）4（每个智能体都有不同的观点）。该测量值反映了团队中信念不一致的程度。环境的动态性越高，信念计数平均值则越大，这是由于智能体不能在决策之前快速达成共识，另外有关环境中动态变化元素的感知信息还应传递给无法直接观察该元素的智能体。在机器人足球中，这些元素包括足球、对手以及团队成员的位置。

在接下来的实验中，每个智能体的审议频率设为 30H_Z，最大通信频率（f_{max}）为 15 H_Z，最小通信频率（f_{min}）为 5 H_Z。对于具有冲突消解的实验，触发冲突消解的周期个数设为 4，权威时间间隔的范围为 $t_{min}=0.8s$ 到 $t_{max}=5s$ 之间。为在实验中进一步引入动态性，仿真器每 5s 随机地重新定位球的位置。

在第一个实验中，仿真器在理想感知器下进行模拟，即没有量测噪声和误差。图 10.7 中给出了丢包情况下的实验结果，图 10.8 给出了人为包延迟的实验结果。其中，每个数据点代表 45min 的仿真结果。

由图 10.7 可得出结论：首先，从图中可知在理想网络情况下且没有冲突消解时，TTC 大约是 100ms，BC 值为 1.04。因此，在平均经过一个通信周期和三个审议周期之后，机器人会达成共识。由于在该配置下没有持续冲突，因此有冲突消解的任务分配的性能相同。尽管在 Kaminka 和 Tambe[82] 的实验中，团队成员在每个审议周期内能够通信，但本实验的结果仍可与其所测得的 ATA

(3.65 tick）相媲美。

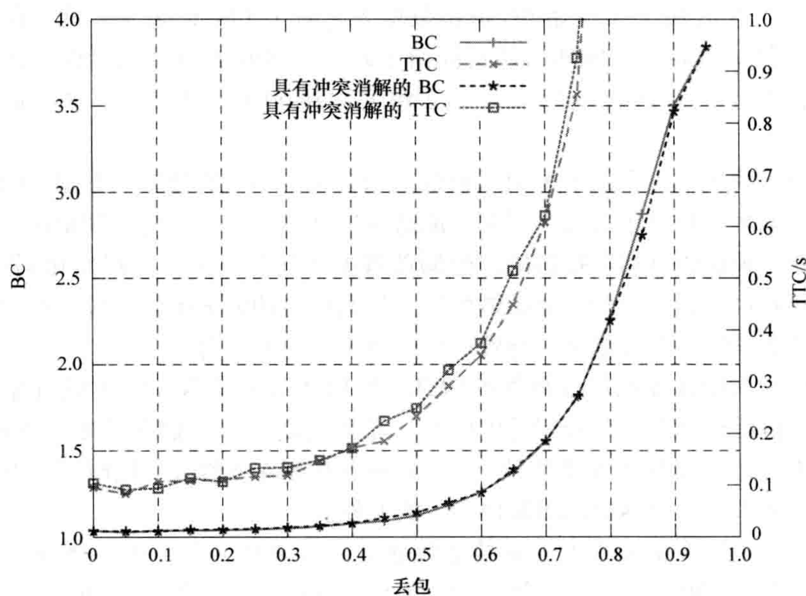


图 10.7 模拟丢包情况下的协调时间和信念状态个数

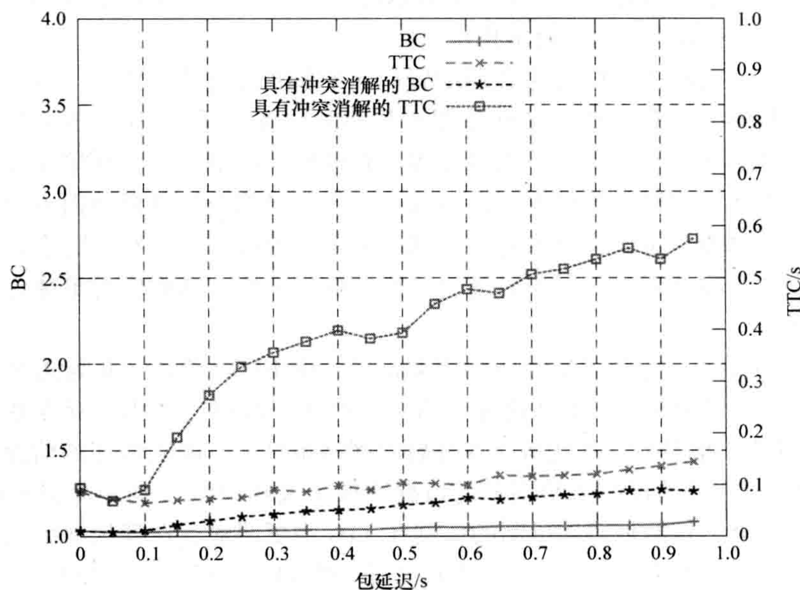


图 10.8 模拟包延迟情况下的协调时间和信念状态个数

其次，在有/无冲突消解的任务分配中，丢包率达到50%之前BC值都是缓慢增大，从达到50%开始BC值指数增大。对于TTC值也是如此。值得注意的是，丢包率越高，智能体就越可能由于缺少所接收的消息而错误地认为其他智能体已损坏（参见7.5.2节）。当丢包率达到60%～70%时，这种情况就会更明显。

相对于丢包率，在系统中引入包延迟会有完全不同的效果，如图10.8所示。无冲突消解时，相对于丢包，团队更能适应包延迟。这也正是所期望的，因为机器人主要是根据仿真器所提供的理想局部数据来进行决策，并得出相同结论的。因此，随着包延迟，指标只会稍微增加。然而在有冲突消解时，包延迟对协调时间具有显著影响。综合起来，导致产生这种影响的原因有两个：

● 由于智能体接收过去事件的信息，然后通过动态重分配来进行相应的更新，因此包延迟会在任务分配历史中引入循环周期，由此触发了冲突消解协议。利用同步时钟（即利用NTP协议[101]）的特点并删除或修改一段时间之前的消息可消除这种影响，由此提高团队协调一致的性能。

● 冲突消解在很大程度上需要依赖通信。权威模式下，由广播分配结果的单个机器人来求解任务分配问题。权威消息的延迟会对团队的协调一致产生很大影响。领导者机器人每次作出决策时，在团队接收并对相应消息做出反应之前，其他机器人都不能达成一致，除了包延迟时间之外，该过程需要50ms。而在正常模式下，智能体可同时对环境作出反应。

在下一个实验中，修改仿真器配置以在感知的足球位置处引入系统误差，使得每个机器人所观测到的足球位置比其实际位置近了30cm。该系统误差大致可模拟在灯光较差的条件下图像处理过补偿的实际情况。在环境光或漫反射光较少而主要是直射光的情况下，造成足球的下半部分呈现黑色而不再被认为是足球的一部分，由此会导致对与足球之间距离的过估计。在实际中，可利用启发式方法消除该影响，但同时又会对该影响过补偿，从而使得足球比实际上显得更近。

图10.9给出了包丢失情况下的协调结果，图10.10给出了包延迟情况下的协调结果。数据表明，当团队内部在信念上产生分歧时（如由于存在系统感知误差），冲突消解具有很大优势。在理想的网络条件下，团队内部的信念状态平均为1.2，这比之前没有系统误差的情况降低了大约20%。利用冲突消解，BC值可降到1.08。这两种协调机制的差别在TTC中更明显，在理想网络状态下利用冲突消解机器人团队在信念上达成共识所用的TTC时间平均为150ms，而冲突持续时间为450ms。在具有理想感知数据的情况下，冲突消解不可能得到如此好的结果，这是由于需要在冲突解决之前首先检测出冲突并进行通信。

直到丢包率达到60%时，冲突消解才能提高性能，若在此时丢失权威消息，

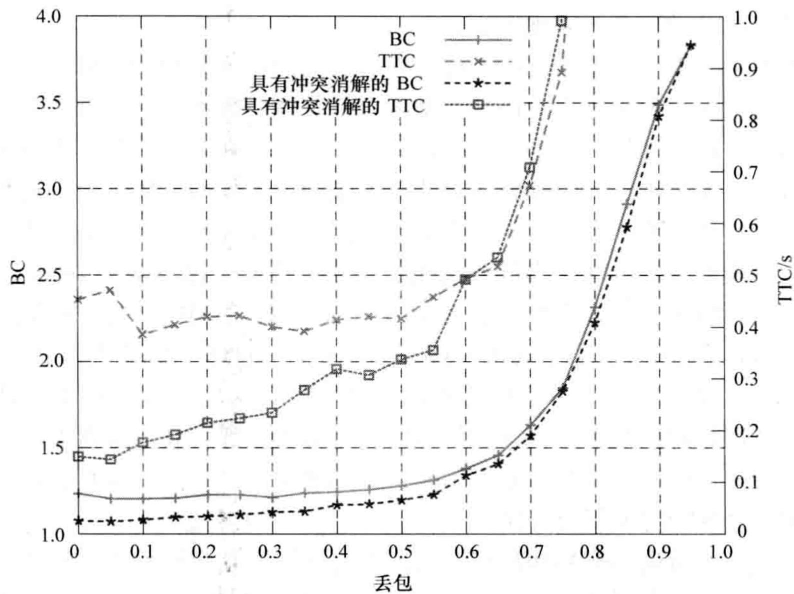


图 10.9 具有系统误差的丢包情况下的协调

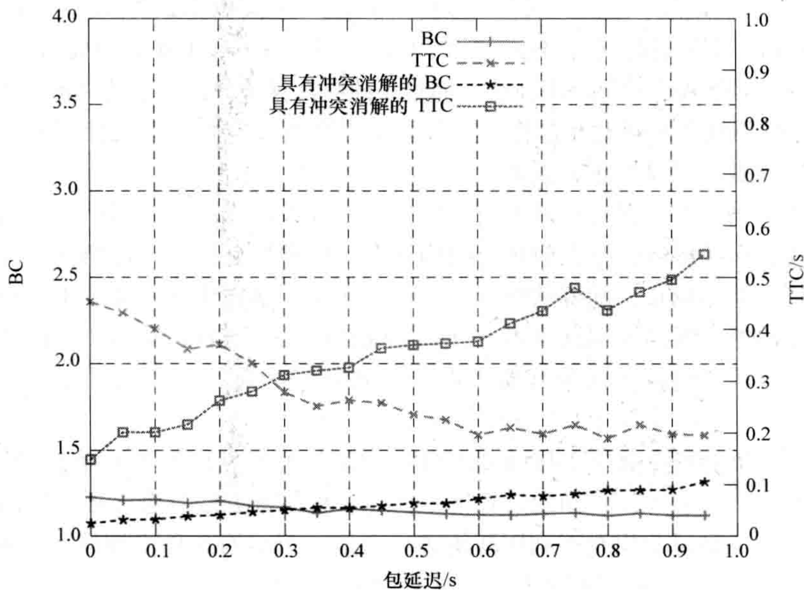


图 10.10 具有系统误差的包延迟下的协调

则会导致更多难以解决的冲突。对于包延迟，也存在类似的情况（见图 10.10），直到延迟达到 300ms 时，冲突消解才会降低协调时间。此时通过通信，团队在

无冲突消解的情况下开始表现得更好。

有趣的是，在本实验中实际上会由于包延迟而减少无冲突消解的 TTC。这是由于感知数据融合和包延迟之间的非常复杂的关联所导致的，包延迟会增大局部观测的运动目标和远程观测的运动目标之间的距离。在某些点处，不再融合不同的观测数据，而是由感知数据融合算法认为是两个不同的物体。由于远程定位信息与本地障碍信息不匹配，由此一个带球机器人会被队友认为是对方球员。另外由于场上只能有一个足球，机器人只能选择一种假设。这可基于 Dempster-Shafer 理论[149]进行精确计算而获得（参见参考文献［131］中有关该场景的详细描述）。在这种情况下，机器人更倾向于离足球最近的那个机器人（即带球机器人）的观测结果，这是由于该机器人具有最高的可信度。因此只要与足球之间的距离、延迟时间以及带球机器人的速度均足够大，那么其他机器人就会采用带球机器人所发送的延迟信息，而不用其自身的感知信息，因此包延迟可部分地提高团队中信念基的协调一致程度。

在此不详细讨论具有感知噪声情况的实验结果。在实验中，感知噪声会导致性能略微下降，这是可想而知的。除此以外，TTC 和 BC 的整体表现与具有理想数据的情况相似。

总之，在理想条件下争议时间为 100ms，50% 丢包率的情况下争议时间为 230 ~ 250ms，而延迟时间为 200ms 下的争议时间为110 ~ 270ms，由此可得出结论：智能体不仅能够快速响应环境变化，而且还能够在任务分配上迅速达成一致，然后按照所建模的规划来行动和合作。此外只要网络条件不太差，就能够通过冲突消解来有效地抵消掉系统感知误差。

在真实场景中，网络质量不可能像上述实验中那么稳定。实际上，网络情况易受突发错误的影响，会在较短时间内影响丢包和包延迟。在此情况下，如果突发错误持续时间较长，则团队性能会如上述实验中那样快速下降。由于每个智能体会计算其独立参与的团队决策，并一直保持该信念，直到获得冲突信息或由于长时间没有接收消息而认为其他机器人已退出该行为，这时就可补偿短时突发错误。

协调一致的程度取决于域的动态性、网络的可靠性以及感知信息的一致性。在这三种因素要求极其苛刻的设置下，任何一种协调方法都会失败，然而利用其他协调方法（如行为识别或利用环境进行通信）可应对这种极端情况。根据 Huber 和 Durfee[75]的工作思路来集成行为识别是今后的研究工作。

10.3 约束求解与优化

对于多智能体协作，ALICA 区别于其他方法的主要特点之一是集成了约束

满足和优化问题。在定义 9.1 中，提出了一个适用于各种机器人和多机器人的问题类，即非线性连续约束满足问题（CNLCSP）。随后，设计了一个能够在真实场景中解决该问题类并且协调团队结果的求解器。该求解器以及问题类的目的是提供一个具有可交换性的相应接口，由此对于不同的域，可利用不同的问题类来描述团队行为，然而上述算法已集成到 ALICA 的参考实现中。在参考文献［155］中利用机器人领域中所提炼的问题，对该求解器以及其他求解器的性能进行了详细评估。在这些实验中，第 9 章所介绍的求解器性能最佳，能够在可接受的时间内以近乎实时推理的方式求解大多数基准约束问题。

接下来的一系列实验对抗噪稳定性以及利用分布式约束满足和优化方法所实现的协调程度进行了评估。在每个实验中，智能体团队求解一个类似于机器人场景所发生的约束满足或约束优化问题，每个参与的智能体都单独具有高斯噪声。类似于真实场景，每个智能体在每个审议周期内都求解约束问题。经 500 次迭代后，智能体终止运行。

最后，测量随时间变化的每个智能体的标准偏差以及任一时间点处团队的标准偏差。

10.3.1　约束环问题

首先，以一个简单的二次优化约束问题来评估所提方法。

设 c 为 $[-10^4, 10^4]^2$ 中的一个固定随机点，寻找一点 $p \in [-10^4, 10^4]^2$ 以使得

$$o(p) = 4 \times 10^4 - |c - p|$$

在 $|c-p| \geqslant 2 \times 10^3$ 条件下最大。

最优解集合是以 c 点为圆心，半径为 2×10^3 的一个圆，该圆外的所有点都为解。值得注意的是，其中的常数以及目标函数是用于保证函数映射到大于 1 的值。这是由于求解器中的一个实现细节，仅是通过一个较小的边距来简化必要计算，并没有其他影响。在本实验中，每个参与智能体都在偏差为 σ_{in} 的各向同性高斯噪声下来感知点 p。

图 10.11 给出了随时间变化的每个智能体的平均标准差。注意到，在只有 1～2 个智能体的情况下，这种输出噪声会急剧增大。而噪声较大时，智能体就无法可靠地跟踪之前的解。然而随着智能体参与数量的增多，噪声会明显下降。若增加第三个智能体，这种效果会更加明显，之后呈现出收益递减的效应。这种情况正如所预期的那样，交换解作为局部搜索的起始点，只要存在大多数，则多数表决结果会得到稳定解。增大表决投票的数量对解的影响非常小。

图 10.12 给出了团队在任一时间点的平均标准差，这是对团队协调一致程度的一种衡量。噪声测量值越高，则团队行为就越不能协调一致。通过两种测量方法的比较，可知团队噪声始终小于随时间变化的噪声。因此尽管作为整体的团队

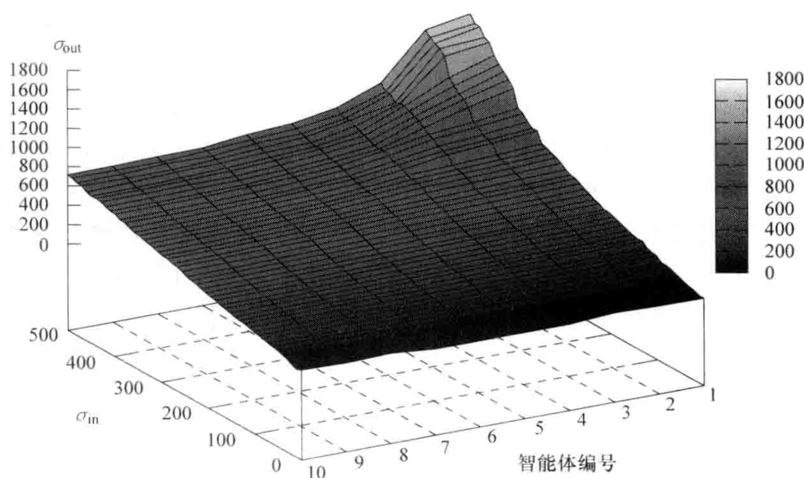


图 10.11 约束环问题中随时间变化产生的噪声

仍然易受到感知噪声的影响，但还是能够达到协调一致。实际上，团队噪声仅略高于输入噪声。

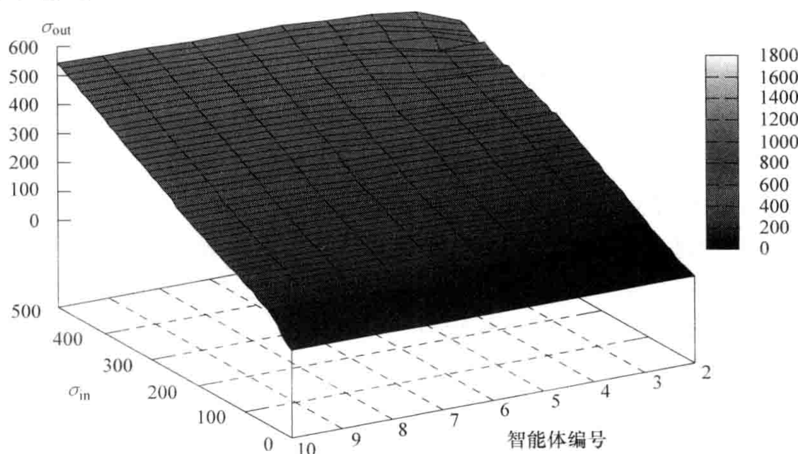


图 10.12 约束环问题中团队所产生的噪声

图 10.13 给出了输入噪声为 $\sigma_{in}=400$ 时由两条曲线组成的二维图像。由图可知，当增加第三个智能体时，噪声水平随时间变化急剧下降，而且对于新增加的智能体，收益递减更加明显。此外增加智能体后，团队噪声只会稍微增加。这些噪声的增大反映了由于输入点 c 的分布而导致的解的分布。

10.3.2 拦截问题

在第二个实验中，采用机器人足球领域中的一个简单问题。给定四个对手机

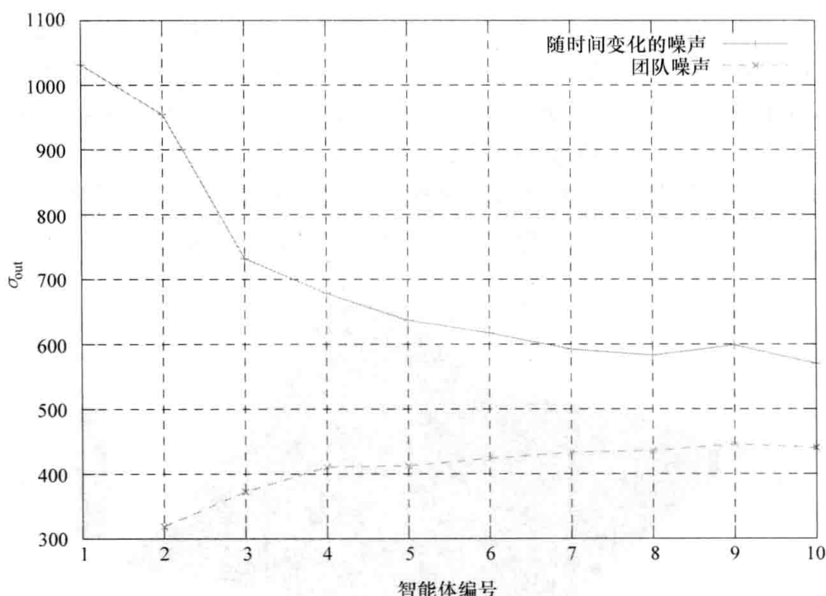


图 10.13 约束环问题中 $\sigma_{in}=400$ 时所产生的噪声

器人的位置以及足球位置，任务是确定三个场上位置，分别是至少离球 2m 远、机器人之间的相互距离至少 1m 以及在对手与球的方向上距离 0.7m 远。这是一个无目标函数的析取约束满足问题。

一般而言，给定$[-10^3\text{mm}, 10^3\text{mm}]^2$中某一固定随机点 b，确定 p_1，p_2，$p_3 \in [-10^4\text{mm}, 10^4\text{mm}]^2$，且 o_1，o_2，o_3，o_4 为距离 b 点 2.5~ 8m 范围内均匀分布的固定点。则解必须满足：

$$((\forall i)\,|p_i - b| \geqslant 2\text{m}) \wedge \tag{10.1}$$

$$|p_1 - p_2| \geqslant 1\text{m} \wedge |p_1 - p_3| \geqslant 1\text{m} \wedge |p_2 - p_3| \geqslant 1\text{m} \wedge \tag{10.2}$$

$$\left((\forall i)(\exists j)\left|p_i - o_j + \frac{0.7\text{m}\cdot(b = o_j)}{|b - o_j|}\right| < 0.01\text{m}\right) \tag{10.3}$$

式（10.3）中下标 i 和 j 分别表示相应的结合与分离的缩写。由此，解限制为四个对手机器人之前，精度为 1cm 的任意点组合。通常，会对该场景增加一个目标函数以使得每个机器人运动到目标位置的距离最小。在此，没有设置目标函数，是为了保证这是一个纯粹的约束满足问题，并且可避免引入目标函数所具有的噪声。所有观测值（包括球和对手的位置）都易受到各向同性高斯噪声的影响，其中偏差为 σ_{in}，测量精度为 mm。

图 10.14 给出了随时间变化所产生的噪声，这与 10.3.1 节中比较简单的结合约束优化问题具有相同特征，只是所产生的噪声更大。这是可预见的，因为在

方程中不只一个噪声点，而是有五个噪声点，且每个可能的目标位置都与两个噪声点有关。另外，一旦智能体无法跟踪解，就可能切换到一个完全不同的解（即分离中的不同情况），从而产生一个较大的方差。在某些情况下，由于第二个约束要求目标位置之间的距离最小，因此甚至必须跟踪一个解。在某些情况下，CSP 甚至是不可解的，即如果对手机器人之间相互距离过近或对手机器人与球距离过近。这种情况发生的概率为 0.042%，即每 45s 发生一次。

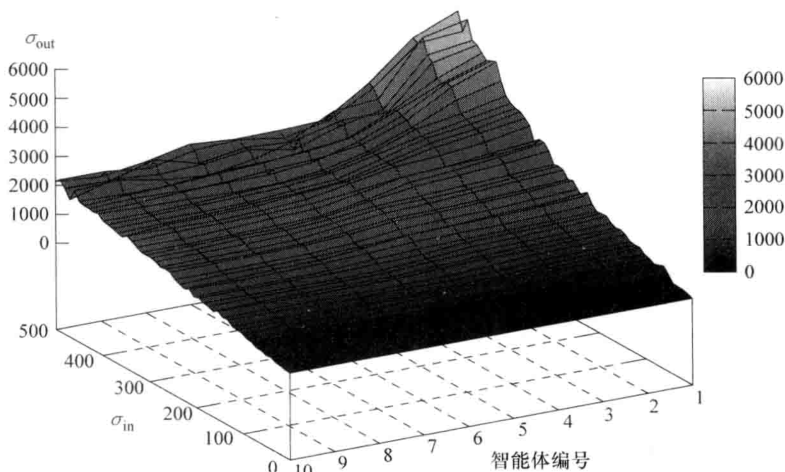


图 10.14　拦截问题中随时间变化所产生的噪声

同理，图 10.15 给出了与上一示例相同的趋势。团队所产生的噪声总是低于随时间变化的噪声，因此即使在丢失对解的跟踪时，机器人也能够协调一致。图 10.16 给出了 1 ~5 个智能体随时间变化所产生噪声的二维投影。再次表明，一旦有第三个智能体参与，则噪声会急剧下降，且每增加一个智能体，收益会递减。

10.3.3　逆运动学

最后一个实验是利用逆运动学问题。在此，目标是定位一个 Kinova 生产的 Jaco™机器臂的末端执行器。该机器臂具有六个自由度。在齐次坐标系中对应于点 p 的末端执行器位置为

$$\boldsymbol{p}=\boldsymbol{M}_1\cdot\boldsymbol{M}_2\cdot\boldsymbol{M}_3\cdot\boldsymbol{M}_4\cdot\boldsymbol{M}_5\cdot\boldsymbol{M}_6\cdot(0,0,0,1)^{\mathrm{T}}$$

式中　$\boldsymbol{M}_i$——机械臂中第 i 个关节的一个变换矩阵。

每次变换都与一个自由度有关。末端执行器的朝向由具有元素 m_{ij} 的变换矩阵 $\boldsymbol{M}=\boldsymbol{M}_1\cdot\boldsymbol{M}_2\cdot\boldsymbol{M}_3\cdot\boldsymbol{M}_4\cdot\boldsymbol{M}_5\cdot\boldsymbol{M}_6$ 来定义。在给定目标点 g 和元素为 r_{ij} 的目标旋转矩阵 $\boldsymbol{R}$ 的条件下，目标函数为

$$U=1000-|g-p|-\sum_i\sum_j(m_{ij}-r_{ij})^2$$

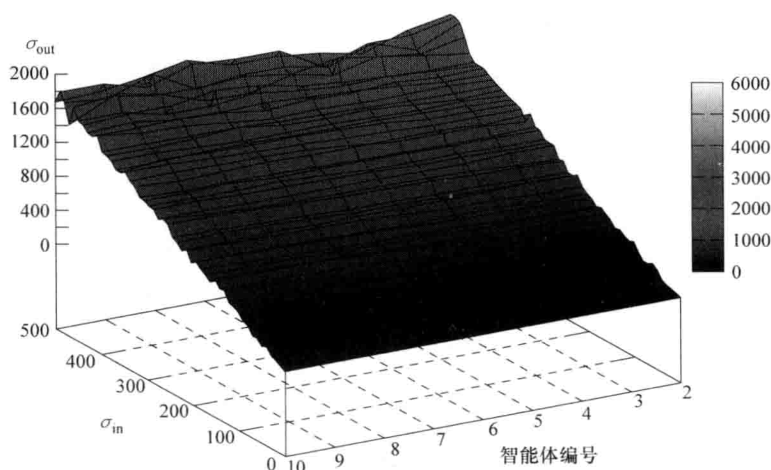


图 10.15 拦截问题中团队所产生的噪声

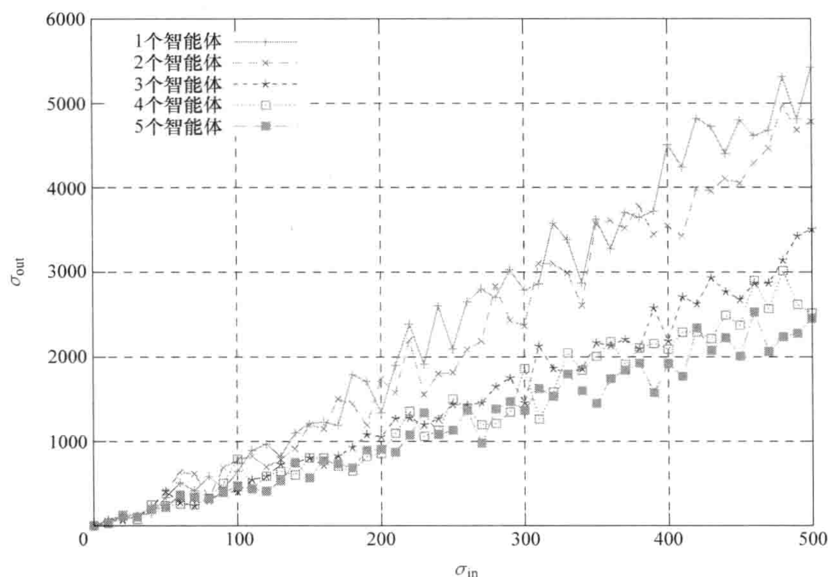


图 10.16 拦截问题中随时间变化所产生的噪声

在此不增加任何约束，因此这是一个完备的非约束优化问题。在实验中，目标点和目标朝向的每一维都受高斯噪声 σ_{in}的影响。

图 10.17 给出了不同个数智能体随时间变化所产生的噪声。值得注意的是，机械臂的最大行程大约是 1.5m，因此相比于机械臂的工作范围，0.1 的输入噪声就已经相当大了，这样噪声较大也就并不奇怪。然而由图可知，由于是多个智能体共同求解，输出噪声会减小。另外，与之前的实验一样，收益递减也非常明显。

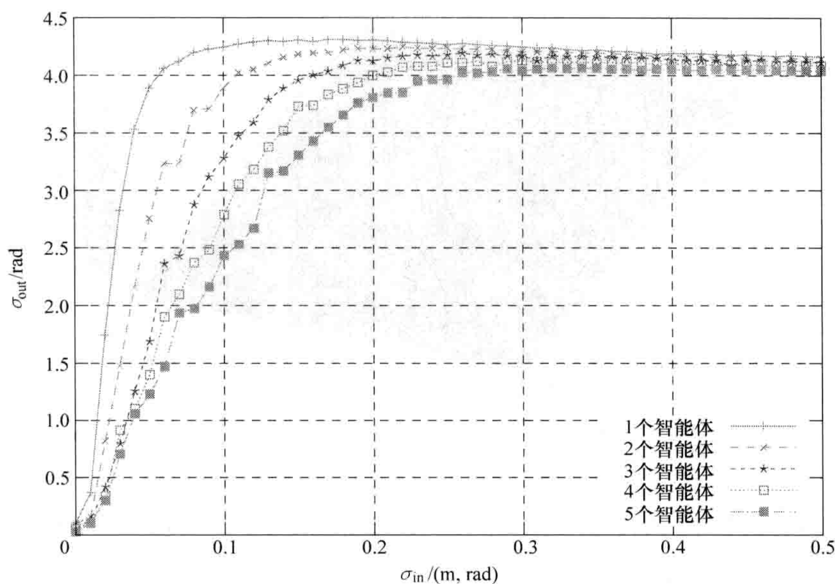


图 10.17 逆运动学约束问题中随时间变化所产生的噪声

为进行比较，在图 10.18 中给出了 5 个智能体时团队产生的噪声和随时间变化所产生的噪声。在具有相同输入噪声的两种测量下，团队产生的噪声要低于随时间变化的噪声。因此在非约束场景下，团队也可以协调一致。

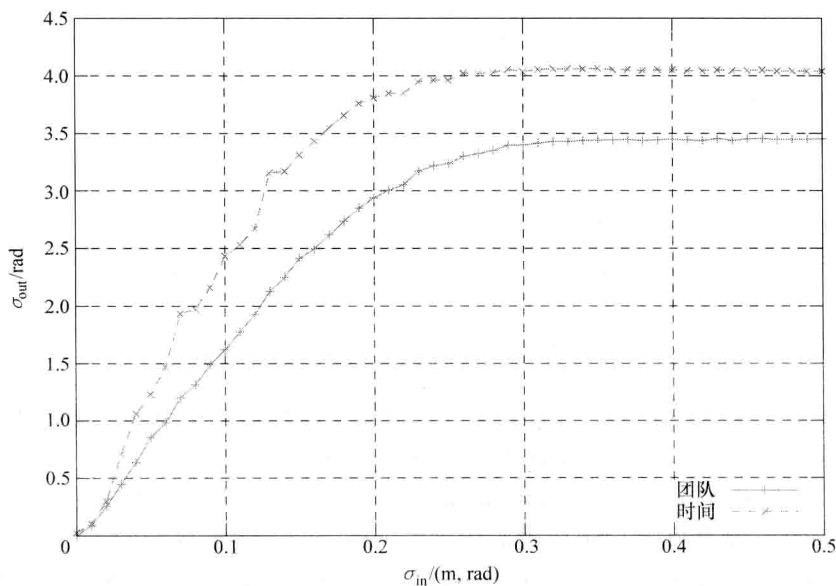


图 10.18 逆运动学问题中 5 个智能体时所产生的噪声

10.3.4 本节小结

总而言之，实验证明，所提算法对于纯粹的约束满足问题、纯粹的约束优化问题以及约束满足与约束优化混合问题均能实现团队的协调一致。对于特定问题，单个决策仍然会在一定程度上受噪声的影响。由于要在反应能力和抗噪鲁棒性之间达到平衡，因此不可能在不牺牲一定程度反应能力的情况下达到更好的结果。在所有实验中，每个智能体都根据具有噪声的局部感知数据求解给定约束问题。也就是说，所用方法主要还是关注于反应能力。值得注意的是，该算法通常与平滑或感知数据融合等预处理相结合。在此，认为这种相结合的方法可应用于所有真实场景。

10.4 案例分析：外太空探索

在 1.3.2 节中，讨论了外太空探索可能是 ALICA 的一个目标场景。由于这种场景通常需要高度专业化的设备来完成特定任务，因此可假设机器人团队都是异构的。即整个团队由不同的机器人组成，每个机器人都具有不同的能力和执行机构。

在 ALICA 中，对异构团队的表示和推理非常简单，这是由于 ALICA 允许根据所需的能力来定义角色，然后在应执行的规划中将这些角色与任务相关联。

考虑一个月球场景如下：一个机器人团队的任务是寻找并取回通信基站的部件，这些部件分布在一个未知环境中。该团队由四个配置有用于探索的不同传感器的小机器人和两个配置有用于抓取和运送部件的机械臂的较大机器人组成。

为区分这两种机器人，引入以下能力：

- 速度：$\{\top, \bot\}$。
- 可抓取：$\{\top, \bot\}$。
- 可运送：$\{\top, \bot\}$。

根据这些能力，可定义团队角色：

$$\mathcal{R} = \{\text{Scout}, \text{Transporter}\}$$

侦察（Scout）角色要求机器人速度快，而运输（Transporter）角色则要求机器人可抓取和运送物体。然后通过角色分配将小机器人与侦察角色相关联，而较大机器人与运输角色相关联。值得注意的是，这个例子稍显简单，不过可以很容易地想象一个更异构的团队组成，其中某些机器人可抓取物体，但不能运送太远。在这种情况下，运输角色就可分解为运输和装载两种角色。

图 10.19 描述了如何在最高层将该问题表示为一个 ALICA 规划。侦察机器人可执行侦察任务，运输机器人执行取回（Retrieving）任务。所有决策都是基于角色和任务的优先级完成，这包括一个特定角色是否能执行一个特定任务，以

及执行的好坏程度。

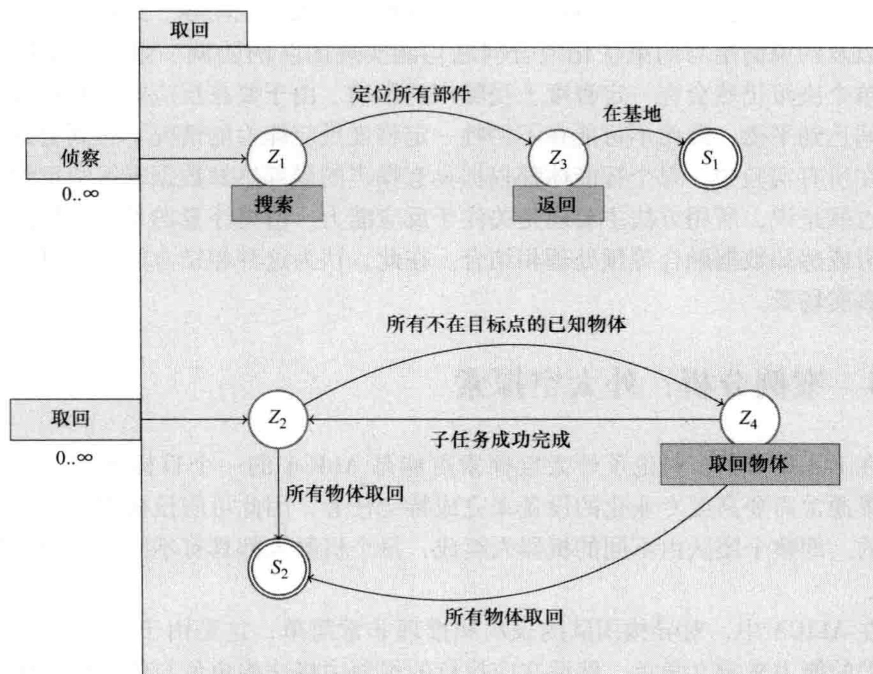


图 10.19 搜索和取回的规划

侦察机器人执行搜索（Search）规划类型，直到所有部件均找到，然后返回基地。运输机器人在状态 Z_2 下空闲，直到发现一个部件，这时开始执行取回物体（RetrieveObject）的任务。一旦完成一个取回物体的任务，则相应的机器人就又会恢复到空闲状态 Z_2。若所有的物体都被取回，则所有的运输机器人就成功地完成该任务。

取回物体和搜索这两个规划类型是进一步展示 ALICA 能力的很好示例。首先讨论取回物体规划类型。正如取回规划的基数所示，该规划类型应工作于任意个机器人和物体的情况下。

10.4.1 取回物体规划类型

在后面的章节中，假设存在以下的域谓词：

- Distance(a, b, d)：d 为位置 a 和 b 之间的距离。
- componentsLocated(n)：n 为已定位但还未被运输的部件个数。
- Position(a, p)：智能体 a 位于位置 p。
- Near(p, o)：位置 p 靠近物体 o。
- Component(o)：o 为一个已定位但尚未被运输的部件。

图 10.20 给出了一种取回物体的可能实现：取回物体实现 RetrieveObjImpl。该实现包括两个任务：取回和空闲。直观上，多余的运输机器人应空闲，直到侦察机器人寻找到更多的部件。这是由运行时条件和效用函数相结合来实现的：

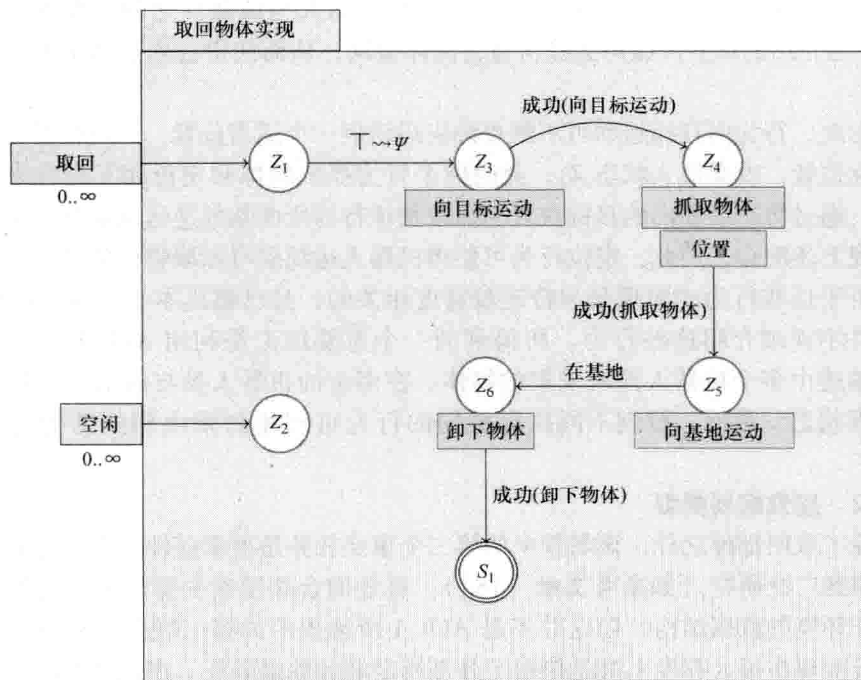


图 10.20　取回物体的规划

$$\text{Run}(\text{RetrieveObjImpl}) = \text{componentsLocated}(n) \wedge |\{a \mid \text{In}(a, \text{RetrieveObjImpl}, \text{Retrieving}, z)\}| \leqslant n$$

$$\mathcal{U}(\text{RetrieveObjImpl}) = \text{pri}(\text{RetrieveObjImpl}) + \frac{|\{a \mid \text{In}(a, \text{RetrieveObjImpl}, \text{Retrieving}, z)\}|}{|\{a \mid \text{In}(a, \text{RetrieveObjImpl}, \tau, z)\}|}$$

值得注意的是，根据任务分配算法，隐式空闲已实现期望行为。在此，采用显式空闲任务以使得效果更加明显。

每个执行取回任务的运输机器人首先应运动到一个特定物体处，然后抓起并运回基地。这种根据物体对机器人的任务分配是由一个约束系统完成的，记为 ψ：

$$\psi = (\forall a)(\exists g)((\exists z)\text{In}(a, \text{RetrieveObjImpl}, \text{Retrieving}, z)) \rightarrow \text{Position}(a, p) \wedge \text{GoalPosition}(a, g) \wedge \text{Component}(o) \wedge \text{Near}(g, o) \wedge \text{Distance}(p, o, d)$$

$(\forall a')((\exists z)\,\text{In}(a',\text{RetrieveObjImpl},\text{Retrieving},z)\wedge a\neq a')\rightarrow\text{Position}(a',p')\wedge\text{Distance}(p',o,d')\wedge d<d'$

式中 GoalPosition(a, g) ——智能体功能流。

ψ 中包括运输机器人接近要运输部件的一个目标位置。另外，还应限制该机器人与所选部件之间的距离要小于其他运输机器人与该部件之间的距离。定义8.8 中所介绍的展开步骤可去除所有智能体量词，从而使得公式更适于进行约束求解。

由此，行为向目标运动可不断查询应运动到一个适当位置。一旦机器人到达该目标位置，就会进入状态 Z_4，此时两个行为抓取物体和定位相互配合来抓取物体。通过将运动朝向的目标位置与抓取物体行为所面临的逆运动学问题相结合来实现上述配合。因此，定位行为可使得机器人运动到可抓取物体的位置。

由于这些行为的实现是与特定域高度相关的，且已超出本书范围，因此在这里并不详细介绍这些行为。所得到的一个重要结论是利用 ALICA 能够很容易地描述由多个机器人同时采集的物体。在多余的机器人参与进来之前都是处于空闲状态。最后，控制不同执行机构的行为可利用约束由相应的变量进行组合。

10.4.2 搜索规划类型

除了取回部件之外，该场景中的第二个重要任务是搜索部件。合作搜索已在过去得到广泛研究（如参考文献［18］）。有效的合作搜索主要是依靠交换和合并表征环境的数据结构，但这并不是 ALICA 所涵盖的内容。因此搜索规划类型的所有实现在很大程度上都是依赖于外部特定域的数据表征。在此并不深入探讨这种表征方法，而是主要关注于如何利用约束来协调多机器人的运动，以实现对某一区域的搜索。

要搜索一个特定物体，应首先提供某些表明物体可能所处位置的先验知识。这些先验知识可表示为 Dempster-Shafer 理论[131,149]中的一个概率函数或信念函数。团队机器人搜索某一区域所采用的搜索模式可适用于该函数的具体实现。如果没有任何相关信息，则概率函数近似为一个均匀分布。在此情况下，如图10.21a 所示的一个搜索模式就可能是一个比较合理的选择。另一方面，如果在某一特定区域的概率函数较密集（如一个正态分布），则一种起始于正态分布中心的螺旋形模式会更适合。

根据这一搜索模式，可通过一个简单扩展的 ALICA 约束系统来构成一个编队，即将约束问题与全状态反馈相关联。在此编队（如一列）可由一个点和一个角度来描述，该点用于更新实现搜索模式的控制器。在随后的约束问题中，控制器可提供相应的下一个点，从而使得机器人沿着所实现的路径运动。

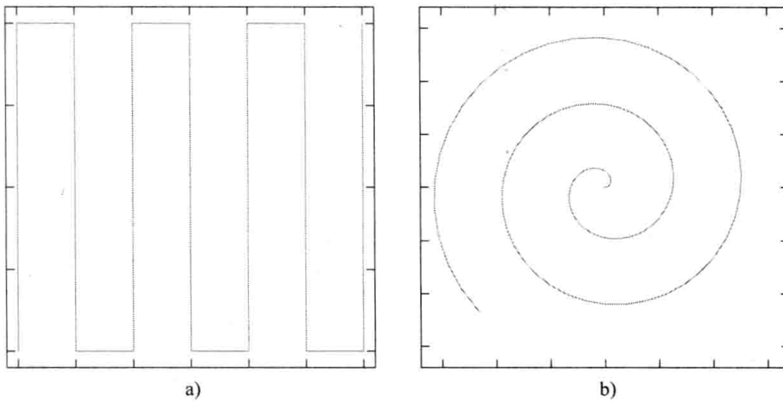


图10.21 搜索模式

a）矩形模式 b）螺旋形模式

本书所提出的架构如图10.22所示。行为不断查询求解器来获得对变量的解，对执行器发布命令并利用新的控制点来更新模式发生器。然后模式发生器调整控制点以适应搜索模式，并产生一个新的控制点，在一个小时间间隔内来更新下一次迭代的约束求解器。实现协调运动的约束问题可由以下宏进行定义：

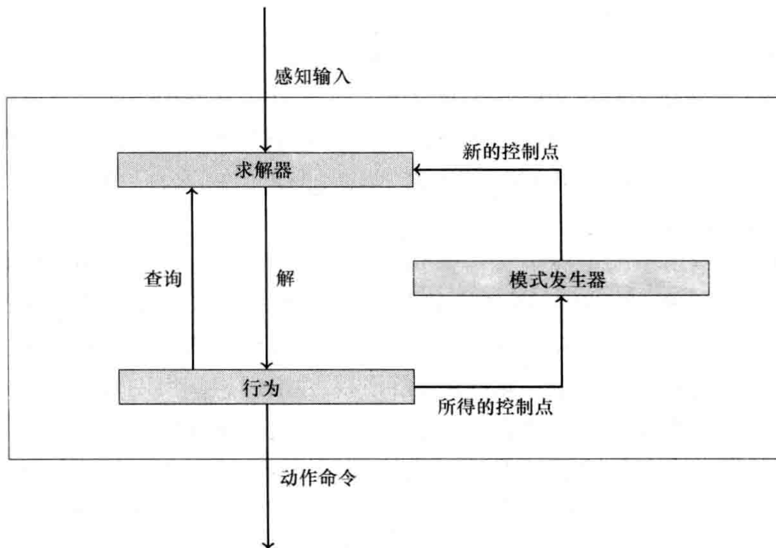


图10.22 编队闭环控制

$$\text{Line}(r,x,y,\alpha,p,\tau,c_x,c_y,\beta) \overset{\text{def}}{=} \sqrt{(c_x-x)^2+(c_y-y)^2}<\text{tol}_d \wedge$$

$$\sqrt{(\cos(\beta)-\cos(\alpha))^2+(\sin(\beta)-\sin(\alpha))^2}<\text{tol}_a \wedge$$

$$n=|\{a \mid \text{In}(a,p,\tau,z)\}| \wedge \text{len}=2rn \wedge$$

$$(\forall a)(\exists z)\text{In}(a,p,\tau,z)\rightarrow(\text{GoalPosition}(a,g)\wedge$$
$$|(g_x-c_x)\cos(\beta)+(g_x-c_y)\sin(\beta)|<\varepsilon\wedge$$
$$|g-c|\leqslant\frac{\text{len}}{2}\wedge$$
$$(\forall a')(\exists z)\text{In}(a',p,\tau,z)a\neq a'\rightarrow$$
$$\left(\text{GoalPosition}(a',g')\wedge|g-g'|\geqslant\frac{\text{len}}{n-1}-\varepsilon\right)$$

式中 c_x、c_y和β——约束变量。

Line（n, r, x, y, α, p, τ, c_x, c_y, β）使得执行规划 p 中任务 τ 的所有机器人的目标位置均匀分布在一条长度为 $2rn$ 的直线上，其中 n 为参与机器人的个数，r 为机器人的观测半径。该直线的位置和朝向由坐标 x 和 y 以及角度 α 确定，使之正交对齐。参数 p 和 τ 分别是指智能体的规划和任务。值得注意的是，该约束充分利用非线性和超越函数来定义相对于控制点（x, y, α）的该直线。

由常量 tol_a、tol_d和 ε 来实现公差。这些公差可对无任何集中控制的 CSP 协调提供裕量。为使得直线中的位置稳定并在机器人的实际位置中集成反馈信息，可对 CSP 增加一个目标函数：

$$o(p,\tau)=|\{a\,|\,\text{In}(a,p,\tau,z)\}|\cdot 10^{12}-\sum_{a:(\exists z)\text{In}(a,p,\tau,z)}|\text{GoalPosition}(a)-\text{Position}(a)|^2$$

$o(p,\tau)$ 可将运动到目标位置的代价作为二次方距离和。

该约束优化问题的解包括两个方面。首先包含所有参与机器人的目标位置，其次还包含一个由 c_x、c_y和 β 给定的控制点，即在允许的公差范围内，最大可能地偏离输入控制点。

图 10.23 给出了四个模拟机器人执行一个矩形模式（见图 10.23a）和一个螺旋形模式（见图 10.23b）所产生的路径。机器人能够协调运动以保持一个搜索路线。值得注意的是，在此并没有采用其他的合作方法。模式发生器的初始状态由约束问题的初始解决定。然而运动的结果并不理想，由图可知，存在一些噪声，且在螺旋形模式中，机器人还可能在编队中偶尔交换位置。

尽管这些结果并不理想，但仍可表明利用 ALICA 中的约束优化问题以及产生搜索模式的一个反馈系统可形成并协调根据给定模式下的搜索路线。该方法仅作为概念验证，还需要进一步研究。虽然这些分析已超出本书范畴，但该方法的优点还是显而易见的：

- 直接与其他编程语言的元素集成，从而可在状态转移的不同行为之间或动态重分配的任务交换之间来回切换。
- 在机器人损坏时可允许自动增强，此时搜索路线自动缩小，从而在搜索区域无间隙。

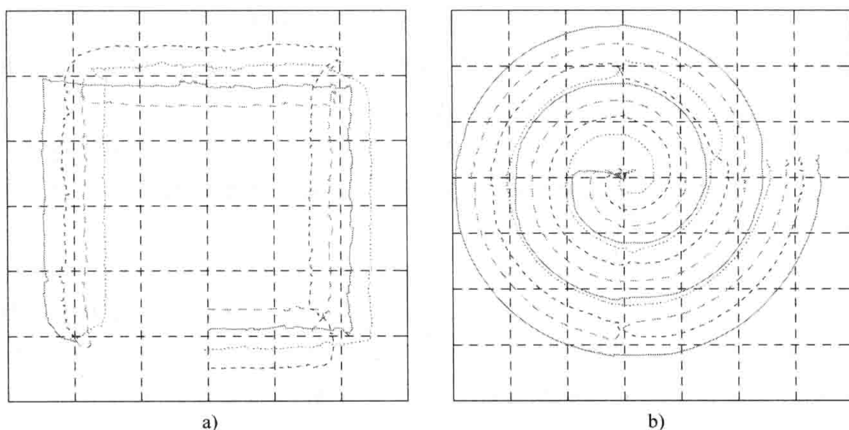


图 10.23　执行搜索模式的机器人
a）矩形模式　b）螺旋形模式

- 最后，该方法具有可进行分析的清晰的数学基础。

10.4.3　本节小结

本节利用 ALICA 对搜索未知区域中特定物体并将这些物体运回基地的团队机器人行为进行了建模，阐述了如何组合 ALICA 中的不同建模元素以形成对团队行为的一个简洁且鲁棒的描述。该建模方法已应用于 DLR 协调项目 IMPERA。

最后，简要描述了如何对基于约束优化的行为描述进行扩展来描述一个机器人动态系统。对于后者的结果非常具有前景，将这些描述与理论相结合，也是今后进一步研究的内容。

10.5　搜救仿真

在最后一个评估场景中，考虑搜救领域。后面将介绍的实验的目的如下：

- 证明 ALICA 可用于其他约束问题领域。
- 评估在其他技术方面的竞争力。
- 评估在团队智能体个数方面 ALICA 的可扩展性。

接下来，采用 Kleiner 等人[86]提出的一个 RoboCup 搜救仿真项目㊀的开源仿真器 RMASBENCH。实验中，模拟发生一次灾难性事件。城市中多处着火，并在建筑物中迅速蔓延，威胁着整个城市。该场景中的智能体作为能够到处运动并灭火的消防队员。仿真截图如图 10.24 所示。

㊀　http：//www.robocuprescue.org。

智能体所面临的主要问题之一是要确定哪个智能体负责扑灭哪处火灾。由于火势蔓延以及被扑灭都是一个动态域，因此智能体应不断地更新决策。RMASBENCH 针对该问题可提供某些集成技术：

- 示范智能体：该策略作为一个基准。每个智能体根据效用函数选择一个附近的目标。由于每个智能体都根据自身来进行决策，因此可看作一个简单的分布式方法。
- 匈牙利指派算法[90]是一种将 n 个智能体最优分配给 m 个任务的集中式算法。最多为每个任务指派一个智能体，这将会造成在很多情况下没有最优解。
- DSA（分布式随机算法）[185]是一种用于分布式约束优化的分散式方法。在给定关于其他智能体的当前信念条件下，每个智能体可计算最佳分配。并以某一固定概率（本实验中取 0.5）应用。如果一个智能体改变其分配，则需将结果向整个团队广播。

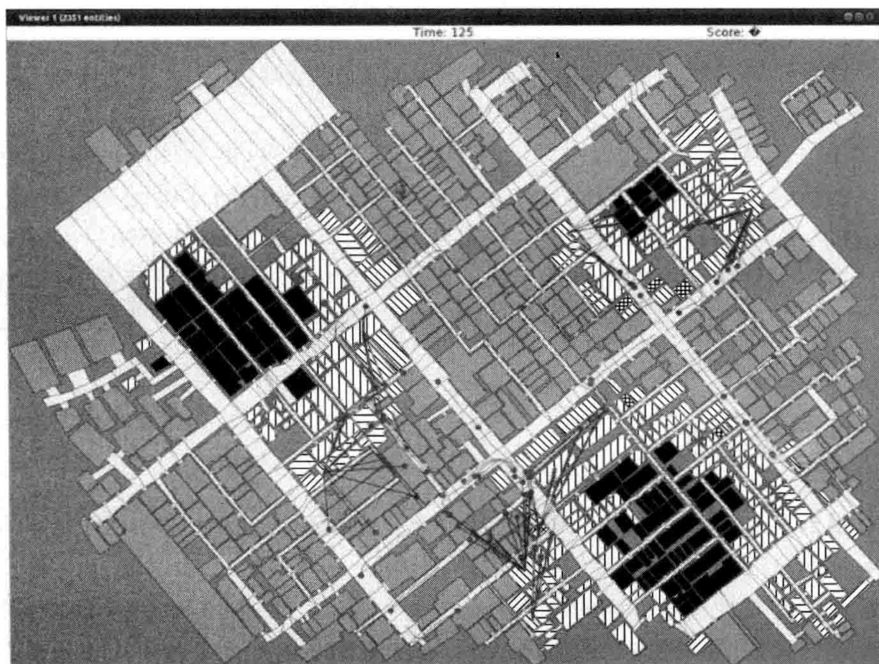


图 10.24　RMASBENCH 截图（各个点表示消防队，不同级别的火势分别由、和的建筑来表示。已扑灭的建筑为和。黑框表示已烧毁）

除了解决分配问题的算法外，问题表征对整个团队的性能也有很大影响。RMASBENCH 中采用的默认问题表征方式是将某一效用值与每个智能体—着火点对相关联：

$$U(a,f)=\begin{cases}0 & \text{如果分配给 } f \text{ 的智能体个数超过一个建筑的特定值 } n(f)\\ \dfrac{10^{10}}{d(a,f)} & \text{如果火势还处于早期阶段}\\ \dfrac{10^{3}}{d(a,f)} & \text{如果火已燃烧得非常厉害}\\ \dfrac{10^{7}}{d(a,f)} & \text{其他}\end{cases}$$

式中 $d(a, f)$ ——智能体 a 和着火点 f 之间的距离。

该效用值对火灾的早期阶段具有重要作用，易于扑灭，而且通常发生在较大火灾的外围。对其中每个智能体为一行，每个火灾点为一列的结果矩阵进行归一化，并作为上述分配算法的输入。

为了将这些方法和 ALICA 进行比较，需要将该效用函数转换为一个适合于 ALICA 概念的值。然而 ALICA 中的任务都是静态的，即一个给定规划中具有固定的任务集合，而着火点的个数则是随时间动态变化的。如果没有一个其他的生成算法，那么该问题就无法表示为一个 ALICA 的任务分配。实际上，该约束问题可利用约束条件和目标函数进行表示。目前所考虑的约束问题类都是采用连续值，而此处的约束问题是离散的，因此需要将给定的离散问题转换为一个连续的约束优化问题。基本思想是对智能体的目标位置进行优化。如果目标位置与着火点之间的距离小于某一阈值，则就考虑分配相应的智能体去灭火。对于该问题，类似于 10.4 节中的实验，再次采用智能体功能流来确定智能体的目标位置。

设 F 为一个着火点集合，$U_s(a, f)$ 表示原始效用函数 $U(a, f)$ 在智能体不多的情况：

$$U_s(a,f)=\begin{cases}\dfrac{10^{10}}{d(a,f)} & \text{如果 } f \text{ 仍处于早期阶段}\\ \dfrac{10^{3}}{d(a,f)} & \text{如果 } f \text{ 燃烧得很厉害}\\ \dfrac{10^{7}}{d(a,f)} & \text{其他}\end{cases}$$

由此可建立一个连续的目标函数 $o(p, \tau)$：

$$o(p,\tau)=\sum_{a:\mathrm{In}(a,p,\tau,z)}\sum_{f\in F}\left(d_g(a,f)<\varepsilon ?\ U_s(a,f):0\right)$$

式中 $d_g(a, f)$ ——相对于智能体 a 和着火点 f 之间的实际距离；

$d(a, f)$ ——a 的约束目标位置与 f 之间的距离；

? ——三元运算符，用于具体化约束条件：

$$(\phi ? a : b) \overset{\text{def}}{=} \begin{cases} a & T(\phi) > 0 \\ b & \text{其他} \end{cases}$$

$$\frac{\delta}{\delta x}(\phi ? a : b) \overset{\text{def}}{=} \begin{cases} -(a-b)\dfrac{\delta}{\delta x}T(\neg\phi) & T(\phi) > 0 \\ (a-b)\dfrac{\delta}{\delta x}T(\phi) & \text{其他} \end{cases}$$

约束条件的具体化是一种避免约束条件产生组合爆炸的常用约束规划方法。在此，定义一个特殊的梯度，以使得求解器中不会出现一个平面场景。

对每个着火点只能分配有限个智能体的要求可用约束 ψ 表示：

$$\psi = \bigcap_{i=0}^{n}\Big(\sum_{a:\mathrm{In}(a,p,\tau,z)} (d_g(a,f_i) < \varepsilon ? 1:0)\Big) \leqslant n(f_i)$$

在此，并不限制将所有的智能体分配给一个着火点，否则可能会导致着火点较少时的一个约束不满足问题。根据约束 ψ 和目标函数 $o(p, \tau)$，可在连续空间下近似一个原始分配问题。每个智能体的目标位置都是笛卡尔空间下的一个二维矢量。通过城市规模大小可推导这些位置的合理边界。

这时一个 ALICA 行为就可查询其目标位置，并对仿真器发布朝最接近该位置的着火点运动并灭火的命令。

ALICA 的规划相当简单。所有智能体执行同样的任务并处于同样的状态。出于评估的目的，将该策略称为 GlobalCOP。针对该策略的 COP 核心公式的规模与着火点个数以及智能体个数成线性关系。另外，搜索空间的大小会随着智能体个数的增加而指数增大。因此认为若智能体超过某一特定数量，则该策略不可行。但由于 ALICA 具有很强的约束问题分解作用，因此利用该特点来评估称为 RegionCOP 的第二种策略。该策略根据分别占地图的 1/4 的四个任务来划分团队。在每个任务中，智能体根据上述类似方法来求解 COP，但需要在 1/4 地图中限制执行任务的智能体以及着火点。为实现任务分配，采用如下的效用加数：

$$f(B) = \sum_{i=1}^{i=4} P_i\Big(0.9 + \frac{0.1}{|\mathcal{A}|}\sum_{a:\mathrm{In}(a,p,\tau_i,z)} (1 - d(a,c_i)^2/\mathrm{maxDist}^2)\Big)$$

式中 $P_i = \min\left(1,\ 1 - F_i + \dfrac{|\{a \mid \mathrm{In}(a,\ p,\ \tau_i,\ z)\}|}{|\mathcal{A}|}\right)$；

$$F_i = \frac{\sum_{f\in M_i} w(f)}{\sum_{f\in F} w(f)};$$

$$w(f) = \begin{cases} 9 & \text{若 } f \text{ 仍在早期阶段} \\ 1 & \text{若 } f \text{ 燃烧剧烈} \\ 4 & \text{其他；} \end{cases}$$

M_i——第 i 象限中的着火点集合；
$d(p, q)$ ——常用距离函数；
c_i——第 i 象限的中心点；
τ_i——与第 i 象限相关的任务。

直观上，$f(B)$ 是测量四个象限中着火点的加权分布，并对智能体也采用类似的分布。另外，智能体首先选择接近的象限区域。由于搜索空间较大（如四个象限中有50个智能体），则如果允许空闲 5^{50}，就会有 4^{50}种可能，且启发式算法必须非常精确。在此采用一种基于贪心分配的简单算法来计算 $f(B)$ 的启发式。

策略 RegionCOP 是采用一种扁平的任务层次结构来简化约束优化问题，同时要求智能体能够立即工作于多个着火点。在接下来的实验中，将采用该策略来验证 ALICA 中任务分配的可扩展性。

所有实验都采用同一场景：在如图10.24 所示的城市中有三处着火的不同位置。在每个实验中，改变智能体个数以及智能体开始工作的时间。智能体工作得越晚，则灭火难度越大，这是因为火情已经开始蔓延。因此在此将起始时间设为 $\max(100, 2n+1)$，其中 n 为智能体的个数。由此可看出，团队规模越大则面临问题的难度也越大。起始时间的上限设为100，以使得在智能体开始行动之前，城市仍保持原样。另外，这个固定的起始时间还可对其他智能体如何采用不同策略来解决同一约束问题进行评估。

在仿真过程的每个时间节拍（tick）中，每个智能体的一个审议周期内包括如下步骤：

- 将来自仿真器的信息集成到环境模型中，这些信息包括城市中智能体的位置和所有着火点的状态。
- 将智能体内部状态的信息及其位置发送给团队。
- 一旦执行一个 ALICA 规则的应用步骤，则所有应用规则都执行。
- 行为不断查询约束的目标位置，并对仿真器发布相应命令。

在每个时间节拍中约束求解器所需的时间设为600ms。注意到这是一个软约束，求解器总是从优先选择的聚类中心开始运行至少一次，以及从随机点开始至少一次搜索运行。本实验是在 Intel® Core™ i7 CPU（2.8GHz）的 Linux3.0.0 平台上运行的。

图10.25 给出了仿真结束时过火后建筑的实验结果，即所有火灾扑灭后，或300个时间节拍之后。图中每个点都是表示十次实验的平均值。值得注意的是，在智能体个数小于40 时，GlobalCOP 与 DSA 算法的性能等效，而若智能体个数大于40，则 GlobalCOP 执行的效果要比 DSA 算法的效果差。在这个场景中，改进 RegioinCOP 策略的性能最佳，尽管在智能体个数较多时可能与 DSA 的性能差别不大，而其他分配策略的性能要差得多。

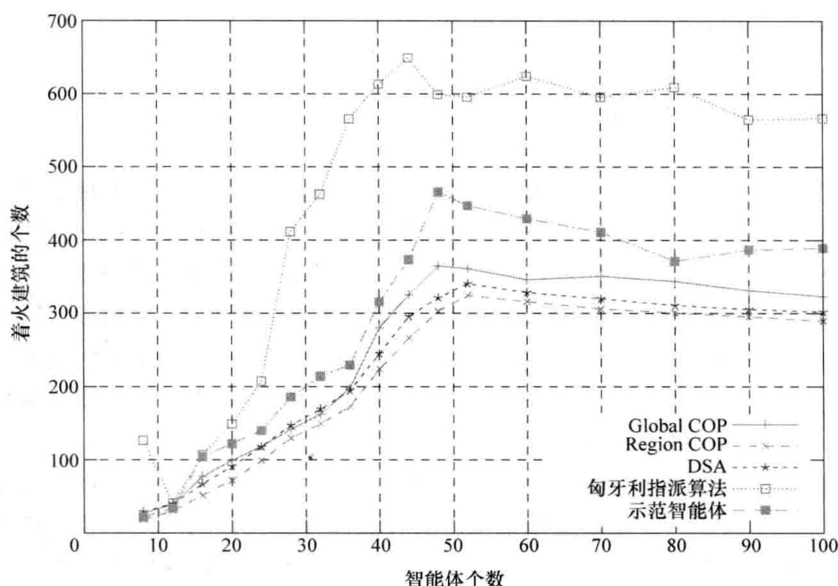


图 10. 25 灭火性能：着火建筑与智能体个数

另外还可看出，由于智能体开始工作有延迟，那么在智能体个数最多 50 个时，所有策略中过火的建筑都会增加。然而对于起始时间固定，具有 50 个智能体且其他条件同上的实验，不管采用什么算法，增加智能体个数只能稍微提高性能。

由此可得出结论，具有 ALICA 求解器的连续原始约束分配问题能够获得相当好的性能。但值得注意的是，DSA 方法要比 ALICA 中连续求解器的计算成本低。这是由于 ALICA 智能体总是考虑整个约束问题，而 DSA 控制的智能体只考虑对局部智能体的选择。另外，由于要映射到连续问题，ALICA 求解器的搜索空间会非常大。本书所提方法主要针对这种连续域，这样的话，如果连续值是必不可少的，那么对于该约束问题会更具可行性。

COP 求解器的可扩展性可由图 10. 26 进行评估。图中给出了每个智能体在每个时间节拍内对于对数标尺上 60 ~ 80 个着火点的环境状态所进行的平均功能评估。评估数会随着智能体个数的增加而急剧下降。这是由于公式规模的增大，问题维度的增大和共享计算功能的仿真智能体个数的增加。当智能体个数较多时，求解器就无法满足 600ms 求解时间的软约束，从而使得功能评估数减少。图中还表明，通过 RegionCOP 对约束问题进行分解可允许在每个时间节拍内执行的功能评估会增加一个数量级，从而使得智能体可以更加彻底地在较小的搜索空间中进行搜索。值得注意的是，图 10. 26 中的功能评估数仅反映了规定时间内的计算次数，而不是分配完成后的计算次数。该求解器花费所有时间来试图寻找一个更

优解，而不管该解是否存在。

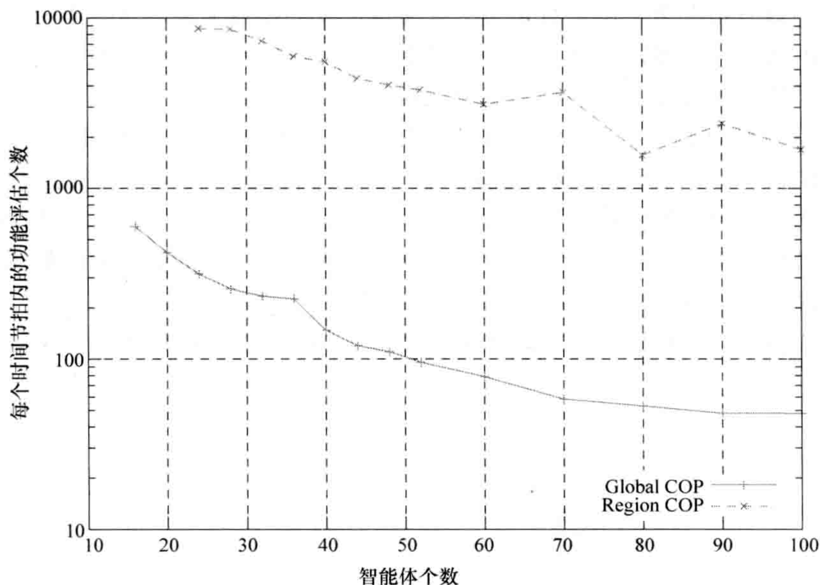


图 10.26　对 60 ~ 80 个着火点的功能评估

当然，提高 RegionCOP 的求解速度并不是没有代价。该策略需要将所有智能体分配给四个任务。图 10.27 给出了如何对该算法进行扩展，图中给出了利用

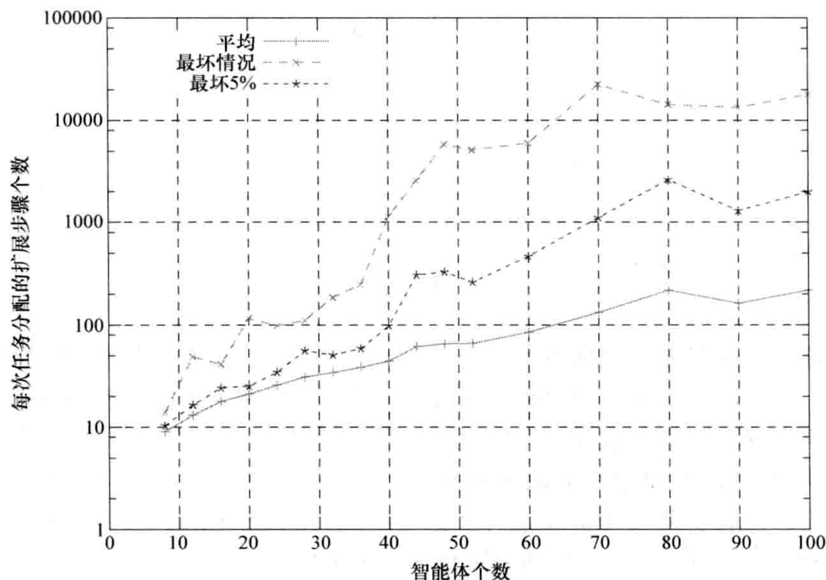


图 10.27　任务分配扩展步骤

A* 搜索算法执行扩展步骤的个数。尽管在平均情况下步骤增多与智能体个数成正比，但最差情况下步骤个数是参与智能体个数的指数关系。这并不奇怪，因为毕竟该问题是一个难解问题。然而为分配 100 个智能体，平均扩展大约 200 次步骤，则算法扩展性相当好。该性能很大程度上取决于所采用的特定启发式方法。

值得注意的是，测试方法的性能会随着应用场景的变化而不同。例如，图 10.28 给出了在不同场景下进行实验时不同算法的性能。实验数据是进行十次实验之后的平均值。RegionCOP 一直保持具有较好的结果，而 GlobalCOP 试图将火势处于可控之下，可在十次实验中四次有效遏制火势。其他所有策略都无法在实验结束时（即经过 300 个时间节拍）扑灭所有的火灾。值得注意的是，在这个特定的实验中，简单的示范智能体策略要比 DSA 策略在表示对场景的影响方面结果更好。

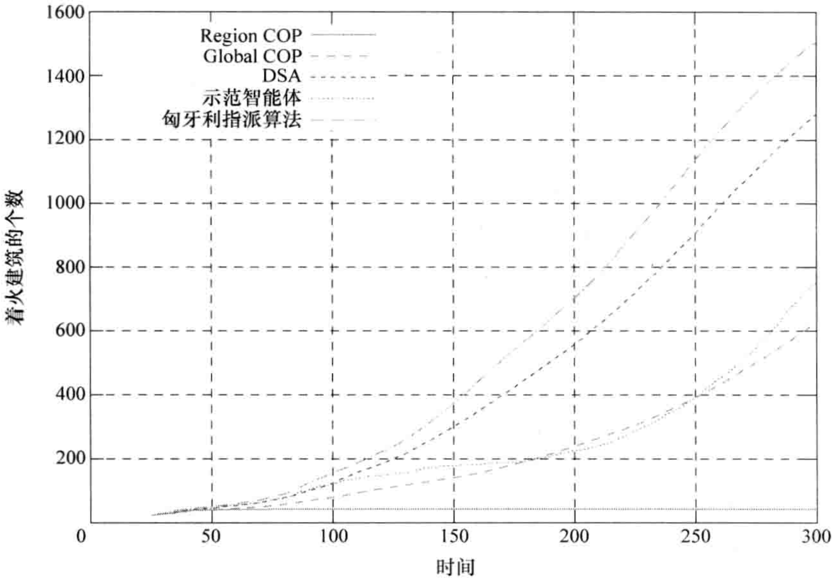


图 10.28　18 个智能体灭火性能示例

总之，在此对搜救领域进行了评估，利用 ALICA 最多可使 100 个智能体协调工作。通过将一个离散的约束优化问题转换到连续域，表明所得的约束问题可利用本书所提方法进行求解。而且相应的策略在性能上可与分布式随机算法处理最多 40 个智能体的原始约束问题相媲美，而同样条件下 ALICA 的性能相对较差。最后，还表明利用 ALICA 语言元素（如任务和效用函数）可实现一个相当简单的约束问题分解，并证明策略要优于原始策略。

第 11 章　总　　结

本书提出了一种对自主机器人团队行为建模的全面解决方法。该方法主要是面对动态环境域，在该领域下，机器人在协调一致行动时必须具有高度的反应能力。

本书的主要贡献在于提出了一种从全局感知角度描述团队行为的编程语言。该语言的语义定义了一个执行层，并进行了详细阐述。从实体机器人或智能体中通过能力将其映射到相应角色来进行抽象，然后又利用角色将机器人动态地分配到规划中的任务中，这些规划是编程语言的核心。通过层次化的有限状态机来描述处理问题的策略或方法。执行层具有一个有效的递归任务分配算法，每个智能体根据该算法来局部计算相关的任务分配。通过暂时及局部地将决策协议自适应切换到一个集中式分配算法来可靠地检测和消解冲突。丧失行为能力的机器人可完全由团队进行补偿。

在本书的第 3 部分，通过采用连续域非线性约束满足和优化问题来表示团队的复杂意图，由此实现对该编程语言的扩展。在运行过程中，参与机器人随时合作利用求解器对上述约束问题进行求解。这些求解器之间的合作重点在于提高单个机器人的反应能力，并在感知噪声情况下实现团队机器人的协调一致性。

所得的框架是完全分布式的，并由有限状态机、基于效用的决策和约束规划的全新组合构成。这种组合可产生一种针对系统设计师的功能强大的建模语言以及规划器等产生式算法。

在理论方面，得到了为实现无冲突分配，产生规划必须满足的可证明条件的层次化任务分配结果。在此给出满足不同需求的两种任务分配算法：一种算法是允许智能体被动参与到一个规划；另一种算法是要求所有智能体在任何时候都主动执行某一任务。

参考实现是一种开源且成功应用于 RoboCup 队伍 Carpe Noctem 的程序。另外，还应用于 DLR 合作项目 IMPERA。

第 10 章中的评估表明 ALICA 可在不同领域（如机器人足球、搜救和外太空探索）成功地控制与协调机器人。另外，实验还展示了对感知噪声和非可靠性通信的鲁棒性。最后，搜救领域的实验结果表明该集成算法的性能可与目前先进的任务分配方法相媲美，利用 ALICA 可控制和协调大型机器人团队。

通过 ALICA 程序的层次化结构以及局部性原理，智能体求解器之间的合作和实现任务分配算法的启发式功能，可实现对 ALICA 的扩展。然而对于过于庞

大的智能体集合，可能需要其他方法（如社会法制或规范）来进行精确建模。对于一种有组织的集成范式，所提出的基于约束的建模方法是一个非常有前途的基础。

11.1 需求分析

在1.2节中，介绍了一组要求所提方法能够处理的具有挑战性的领域。在此，简单总结一下如何解决这些问题。

连续变化的环境 ALICA中的每个智能体都根据自身的局部决策进行动作，无需任何事先通信，由此可对所有未知变化直接反应。动态重分配可允许智能体切换任务或整个规划来适应这种动态变化。另外，约束求解器也总是针对当前状态来求解一个按需构建的约束问题，因此也重点关注于反应能力。在实现方面，是利用规则应用部件来完成的，即在程序层次结构中可随时进行更新而不存在任何由于线程同步所造成的延迟。

具有噪声的感知器 尽管ALICA中没有一个感知融合部件，并且还应与相应的部件配合使用，但在具有一定程度的反应能力时能够具有抗噪功能。首先，在具有噪声的情况下可通过阈值或类似的值来使得任务分配保持稳定。其次，如10.3节所述，团队能够在不牺牲单个智能体反应能力的情况下得到约束问题的稳定解。

部分可观测的环境 作为一个执行和协调层，ALICA并不直接处理该问题。该问题的一种解决方法是部分表示该环境，即ALICA并不参与以保持域的独立性。然而由于ALICA智能体不断交换各自行为信息，整体行为很少依赖于个体信念。这在ALICA所遵循的局部性原理中有着明显的体现，使得智能体并不考虑自身没有主动或被动参与的问题或规划。因此，局部信息已足以使得一个智能体完成其在团队中的任务。

非可靠性通信 网络质量较差情况下的鲁棒性已在10.2节中进行了详细评估。结果表明，在高丢包率或延迟情况下仍具有高度的协调一致性。

团队成员损坏 丧失行为能力的机器人可直接通过动态重分配来补偿。由于缺少能力或机器人，不再执行当前规划，若存在另一个规划，就选择执行该规划。否则就会产生故障，并通过程序的层次化进行传播，直到该问题解决。

11.2 展望与未来

本书工作主要是对自主移动机器人团队行为的建模和协调提出一种全面的解决方案，但仍存在一些问题尚待解决。另外，ALICA还可进行扩展以更便于集

成更多的部件。接下来，对一些最重要的问题以及可能进行的扩展进行总结。

规划集成 在传统意义上 ALICA 没有一个规划部件，然而编程语言的设计主要是依靠一种规划的思想，因此可对合适的语言元素（如终止状态和行为）定义后置条件。在运行过程中并没有利用这些后置条件，但在规划期间具有重要作用。在研究项目 IMPERA 中正在对规划集成进行研究。

基于 SMT 的约束求解和优化 第 8 章在 ALICA 中采用了 SMT 求解器，并大致描述了扩展该求解器具有跟踪和协调能力的可能方法。据现有知识，目前还不存在满足本书讨论的 SMT 求解器，然而 9.3 节中的实验表明基于 SMT 的技术可在性能上得到极大提高。

任务分配与约束求解的融合 如 9.7 节所述，ALICA 智能体在运行时具有两种可能存在的计算成本过高的问题。首先是任务（重）分配，其次是约束求解。如果能针对约束问题进行任务分配，将两者集成或许能产生对行为更精确的描述以及更佳的团队性能。目前只有一种可能方法，即采用来自于条件或效用函数中的当前解。如何解决所产生的复杂性问题仍是一个难题。

利用微分约束问题进行动态行为建模 利用约束问题来设定行为的方法，可很容易地通过一个反馈扩展到本质上是一种随时间变化的微分方程的约束问题。在 10.4 节中，讨论了利用该方法如何在没有一个中心部件或指定初始状态的条件下进行动态编队表示。但仍缺乏一种理论来保证如何控制系统的不同特性（如机器人速度和一致性程度）。针对该系统及其相关的其他方法（如 Jaeger 和 Christaller[77] 提出的 Dual Dynamics 设计机制）是今后的重点研究问题。

CSP 的冲突消解 本书提出的针对约束问题的一致性解决方法目前尚不能扩展到单个约束问题，即每个机器人都认为显著不同，但通过某些变量相互关联。处理该问题需要根据 Petcu[121] 所提出的 DCOP 求解器来提供其他信息。由非线性约束、连续域、动态环境和非可靠性通信组合所产生的其他问题仍没有解决。

参考文献

[1] Till Amma, Philipp Baer, Kai Baumgart, Philipp Burghardt, Kurt Geihs, Janosch Henze, Stephan Opfer, Stefan Niemczyk, Roland Reichle, Daniel Saur, Andreas Scharf, Jens Schreiber, Martin Segatz, Florian Seute, Hendrik Skubch, Stefan Triller, Michael Wagner, and Andreas Witsch. Carpe Noctem 2009. In *RoboCup 2009 International Symposium*, Graz, June 2009. TU Graz.

[2] Krzysztof R. Apt and Mark Wallace. *Constraint Logic Programming using Eclipse*. Cambridge University Press, New York, NY, USA, 2007. ISBN 0521866286.

[3] Matthew Arnold, Stephen J. Fink, David Grove, Michael Hind, and Peter F. Sweeney. A survey of adaptive optimization in virtual machines. In *Proceedings of the IEEE Special Issue on Program Generation, Optimization, and Adaptation*, volume 93, pages 449–466, 2005. doi:10.1109/JPROC.2004.840305.

[4] John Aycock. A brief history of just-in-time. *ACM Comput. Surv.*, 35(2): 97–113, 2003. doi:10.1145/857076.857077.

[5] Philipp A. Baer. *Platform-Independent Development of Robot Communication Software*. Phd thesis, University of Kassel, Kassel, 2008. URL http://www.upress.uni-kassel.de/publi/abstract.php?978-3-89958-644-2.

[6] Andreas Bauer, Markus Pister, and Michael Tautschnig. Tool-support for the analysis of hybrid systems and models. In *Design, Automation and Test in Europe (DATE)*, pages 924–929, 2007.

[7] W. Beaton and J. d. Rivieres. Eclipse Platform Technical Overview. Technical report, The Eclipse Foundation, 2006.

[8] R. E. Bellman. *Dynamic Programming*. Princeton University Press, Princeton, N.J., 1957.

[9] Roger Bemelmans, Gert Jan Gelderblom, Pieter Jonker, and Luc De Witte. Socially assistive robots in elderly care: A systematic review into effects and effectiveness. *Gerontechnology Journal*, 8(2):94–103, 2010. doi:10.1016/j.jamda.2010.10.002.

[10] Frédéric Benhamou and Laurent Granvilliers. Continuous and Interval Constraints. In Francesca Rossi, Peter van Beek, and Toby Walsh, editors, *Handbook of Constraint Programming*, pages 571–603. Elsevier, 2006. ISBN 978-0-444-52726-4.

[11] J. Bohren and S. Cousins. The smach high-level executive. *Robotics Automation Magazine, IEEE*, 17(4):18–20, dec 2010. ISSN 1070-9932. doi:10.1109/MRA.2010.938836.

[12] Michael Bratman. *Intentions, Plans, and Practical Reason*. Harvard University Press, 1987.

[13] Michael Brenner. A multiagent planning language. In *Workshop on ICAPS*, May 2003.

[14] Gerhard Brewka, Ilkka Niemelä, and Miroslaw Truszczynski. Preferences and nonmonotonic reasoning. *AI Magazine*, 4:69–78, 2008.

[15] Susan S. Brilliant and Timothy R. Wiseman. The first programming paradigm and language dilemma. In *Proceedings of the twenty-seventh SIGCSE technical symposium on Computer science education*, SIGCSE' 96, pages 338–342, New York, NY, USA, 1996. ACM. ISBN 0-89791-757-X. doi:10.1145/236452.236572.

[16] Kenneth N. Brown and Ian Miguel. Uncertainty and Change. In Francesca Rossi, Peter van Beek, and Toby Walsh, editors, *Handbook of Constraint Programming*, pages 731–760. Elsevier, 2006. ISBN 978-0-444-52726-4.

[17] Wray Buntine. Generalized subsumption and its applications to induction and redundancy. *Artificial Intelligence*, 36:149–176, September 1988. ISSN 0004-3702. doi:10.1016/0004-3702(88)90001-X.

[18] Wolfram Burgard, Mark Moors, Cyrill Stachniss, and Frank E Schneider. Coordinated multi-robot exploration. *IEEE Transactions on Robotics*, 21 (3):376–386, 2005. doi:10.1.1.59.4390.

[19] Carlos Caleiro and Ricardo Gonçalves. On the algebraization of many-sorted logics. In *Recent Trends in Algebraic Development Techniques - Selected Papers*, pages 21–36. Springer-Verlag, 2007.

[20] Adam Campbell and Annie S. Wu. Task and role allocation within multi-agent and robotics research. Technical Report 05, UCF, 2007.

[21] Adam Campbell and Annie S. Wu. Multi-agent role allocation: issues, approaches, and multiple perspectives. *Autonomous Agents and Multi-Agent Systems*, 22(2):317–355, 2011. doi:10.1007/s10458-010-9127-4.

[22] D. Challet and Y.-C. Zhang. Emergence of cooperation and organization in an evolutionary game. *Physica A: Statistical Mechanics and its Applications*, 246(3–4):407–418, 1997. ISSN 0378-4371. doi:10.1016/S0378-4371(97)00419-6.

[23] Stuart Chalmers and Peter M.D. Gray. BDI agents and constraint logic. *AISB Journal Special Issue on Agent Technology*, 1(1):21–40, 2001.

[24] Shyamal Suhana Chandra and Kailash Chandra. A comparison of Java and C#. *Journal of Computing Sciences in Colleges*, 20:238–254, February 2005. ISSN 1937-4771.

[25] Antonio Chella, Massimo Cossentino, Roberto Pirrone, and Andrea Ruisi. Modeling ontologies for robotic environments. In *Proceeding of the 14th International Conference on Software Engineering and Knowledge Engineering*, pages 15–19, 2002.

[26] Xinguang Chen and Peter van Beek. Conflict-directed backjumping revisited. *Journal of Artificial Intelligence Research (JAIR)*, 14:53–81, 2001. doi:10.1613/jair.788.

[27] David Cohen and Peter Jeavons. The Complexity of Constraint Languages. In Francesca Rossi, Peter van Beek, and Toby Walsh, editors, *Handbook of Constraint Programming*, pages 245–280. Elsevier, 2006. ISBN 978-0-444-52726-4.

[28] Philip R. Cohen and Hector J. Levesque. Intention is choice with commitment. *Artificial Intelligence*, 42(2-3):213–261, 1990. ISSN 0004-3702. doi:10.1016/0004-3702(90)90055-5.

[29] Stephen A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the third annual ACM symposium on Theory of computing*, STOC '71, pages 151–158, New York, NY, USA, 1971. ACM. doi:10.1145/800157.805047.

[30] James M. Crawford and Larry D. Auton. Experimental results on the crossover point in random 3-sat. *Artificial Intelligence*, 81(1-2):31–57, 1996.

[31] Nadia Creignou, Sanjeev Khanna, and Madhu Sudan. *Complexity classifications of boolean constraint satisfaction problems*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2001. ISBN 0-89871-479-6.

[32] Robert J. Dakin. A tree-search algorithm for mixed integer programming problems. *The Computer Journal*, 8(3):250–255, March 1965. ISSN 1460-

2067. doi:10.1093/comjnl/8.3.250.

[33] Aniruddha Dasgupta and Aditya K. Ghose. CASO: A Framework for dealing with objectives in a constraint-based extension to AgentSpeak(L). In Vladimir Estivill-Castro and Gillian Dobbie, editors, *Twenty-Ninth Australasian Computer Science Conference (ACSC 2006)*, volume 48 of *CRPIT*, pages 121–126, Hobart, Australia, 2006. ACS.

[34] Mehdi Dastani, M. Birna, Riemsdijk Frank Dignum, and John-Jules Ch. Meyer. A programming language for cognitive agents: Goal directed 3APL. In *Programming Multi-Agent Systems, First International Workshop, PROMAS 2003, Melbourne, Australia*, pages 111–130. Springer, July 2003.

[35] Mehdi Dastani, Dirk Hobo, and John-Jules Ch. Meyer. Practical extensions in agent programming languages. In *AAMAS '07: Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems*, pages 1–3, New York, NY, USA, 2007. ACM. ISBN 978-81-904262-7-5.

[36] Martin Davis and Hilary Putnam. A computing procedure for quantification theory. *Journal of the ACM*, 7:201–215, July 1960. ISSN 0004-5411. doi:10.1145/321033.321034.

[37] Giuseppe de Giacomo, Yves Lespérance, and Hector J. Levesque. Congolog, a concurrent programming language based on the situation calculus. *Artificial Intelligence*, 121(1-2):109–169, August 2000. ISSN 0004-3702. doi:10.1016/S0004-3702(00)00031-X.

[38] Joris De Schutter, Tinne De Laet, Johan Rutgeerts, Wilm Decré, Ruben Smits, Erwin Aertbeliën, Kasper Claes, and Herman Bruyninckx. Constraint-based task specification and estimation for sensor-based robot systems in the presence of geometric uncertainty. *International Journal of Robotics Research*, 26:433–455, May 2007. ISSN 0278-3649. doi:10.1177/027836490707809107.

[39] Wilm Decré, Ruben Smits, Herman Bruyninckx, and Joris De Schutter. Extending iTaSC to support inequality constraints and non-instantaneous task specification. In *Proceedings of the 2009 IEEE International Conference on Robotics and Automation*, ICRA'09, pages 964–971, Piscataway, NJ, USA, 2009. IEEE Press. ISBN 978-1-4244-2788-8.

[40] Gilles Dowek, César Muñoz, and Corina Păsăreanu. A small-step semantics of PLEXIL. Technical Report 2008-11, National Institute of Aerospace, Hampton, VA, 2008.

[41] Markus Eich and Frank Kirchner. Reasoning about geometry: An approach using spatial-descriptive ontologies. In *Workshop AILog 19th European Conference on Artificial Intelligence ECAI10, Lisbon*, 2010.

[42] Ronald Fagin, Joseph Y. Halpern, Yoram Moses, and Moshe Y. Vardi. *Reasoning About Knowledge*. MIT Press, 1995. ISBN 0262061627.

[43] Tomás Feder and Moshe Y. Vardi. The computational structure of monotone monadic snp and constraint satisfaction: A study through datalog and group theory. *SIAM J. Comput.*, 28(1):57–104, February 1999. ISSN 0097-5397. doi:10.1137/S0097539794266766.

[44] K. Feher. *Telecommunications Measurements, Analysis, and Instrumentation*. Noble Publishing classic series. Noble Publishing, 1996. ISBN 9781884932038.

[45] Richard Fikes and Nils J. Nilsson. Strips: A new approach to the application of theorem proving to problem solving. In *Proceedings of the 2nd International Joint Conference on Artificial intelligence (IJCAI)*, pages 608–620, San Francisco, CA, USA, 1971. Morgan Kaufmann Publishers Inc.

[46] FIPA. *FIPA ACL Message Structure Specification*. FIPA, 2001. URL http://www.fipa.org/specs/fipa00061/. (accessed 2012-05-23).

[47] FIPA. *FIPA Communicative Act Library Specification*. FIPA, December 2002. URL http://www.fipa.org/specs/fipa00037/. (accessed 2012-05-23).

[48] Martin Fränzle, Christian Herde, Tino Teige, Stefan Ratschan, and Tobias Schubert. Efficient solving of large non-linear arithmetic constraint systems with complex boolean structure. *Journal on Satisfiability, Boolean Modeling and Computation*, 1:209–236, 2007.

[49] Thom Frühwirth. Introducing simplification rules. Technical Report ECRC-LP-63, European Computer-Industry Research Centre, München, Germany, October 1991. Presented at the Workshop Logisches Programmieren, Goosen/Berlin, Germany, and the Workshop on Rewriting and Constraints, Dagstuhl, Germany,.

[50] Thom Frühwirth. Theory and practice of constraint handling rules. *The Journal of Logic Programming*, 37(1–3):95–138, 1998. ISSN 0743-1066. doi:10.1016/S0743-1066(98)10005-5. URL http://www.sciencedirect.com/science/article/pii/S0743106698100055.

[51] D. Gale. *The Theory of Linear Economic Models*. Economics / mathematics. University of Chicago Press, 1989. ISBN 9780226278841. URL http://books.google.de/books?id=3t3F9rLAZnYC.

[52] J. H. Gallier. *Logic for Computer Science: Foundations of Automatic Theorem Proving*, chapter 10. Many-Sorted First-Order Logic. Harper & Row Publishers, Inc., 1985. Out of print, available via www.cis.upenn.edu/~cis610/logic.pdf.gz (last accessed 10-12-2011).

[53] Harald Ganzinger, George Hagen, Robert Nieuwenhuis, Albert Oliveras, and Cesare Tinelli. Dpll(t): Fast decision procedures. In Rajeev Alur and Doron Peled, editors, *Computer Aided Verification, 16th International Conference, CAV 2004, Boston, MA, USA, July 13-17, 2004, Proceedings*, volume 3114 of *Lecture Notes in Computer Science*, pages 175–188. Springer, 2004. ISBN 3-540-22342-8. doi:10.1007/978-3-540-27813-9_14.

[54] H. Garcia-Molina. Elections in a distributed computing system. *IEEE Transactions on Computers*, 31:48–59, January 1982. ISSN 0018-9340. doi:10.1109/TC.1982.1675885.

[55] Brian P Gerkey. A formal analysis and taxonomy of task allocation in multi-robot systems. *The International Journal of Robotics Research*, 23(9):939–954, 2004. doi:10.1177/0278364904045564.

[56] Jens Gerlach and Joachim Kneis. Generic Programming for Scientific Computing in C++, Java™ and C#. In Xingming Zhou, Ming Xu, Stefan Jähnichen, and Jiannong Cao, editors, *Advanced Parallel Processing Technologies*, volume 2834 of *Lecture Notes in Computer Science*, pages 301–310. Springer Berlin / Heidelberg, 2003. ISBN 978-3-540-20054-3. doi:10.1007/978-3-540-39425-9_37.

[57] Giuseppe Giacomo, Yves Lespérance, Hector J. Levesque, and Sebastian Sardina. IndiGolog: A High-Level Programming Language for Embedded Reasoning Agents. In Amal El Fallah Seghrouchni, Jürgen Dix, Mehdi Dastani, and Rafael H. Bordini, editors, *Multi-Agent Programming Languages, Tools and Applications*, volume 1, chapter 2, pages 31–72. Springer, 2009. doi:10.1007/978-0-387-89299-3_2.

[58] Piotr J. Gmytrasiewicz and Edmund H. Durfee. Decision-theoretic recursive modeling and the coordinated attack problem. In *Proceedings of the first international conference on Artificial intelligence planning systems*, pages 88–95, San Francisco, CA, USA, 1992. Morgan Kaufmann Publishers Inc. ISBN 1-55860-250-X. URL http://dl.acm.org/citation.cfm?id=139492.139503.

[59] Alexandre Goldsztejn, Claude Michel, and Michel Rueher. An efficient algorithm for a sharp approximation of universally quantified inequalities. In Roger L. Wainwright and Hisham Haddad, editors, *Proceedings of the 2008 ACM Symposium on Applied Computing (SAC), Fortaleza, Ceara, Brazil, March 16-20, 2008*, pages 134–139. ACM, 2008. ISBN 978-1-59593-753-7. doi:10.1145/1363686.1363724.

[60] Alexandre Goldsztejn, Claude Michel, and Michel Rueher. Efficient handling of universally quantified inequalities. *Constraints*, 14:117–135, March 2009. ISSN 1383-7133. doi:10.1007/s10601-008-9053-0.

[61] Robin Gras, Didier Devaurs, Adrianna Wozniak, and Adam Aspinall. An individual-based evolving predator-prey ecosystem simulation using a fuzzy cognitive map as the behavior model. *Artificial Life*, 15(4):423–463, 2009. doi:10.1162/artl.2009.Gras.012.

[62] Jim Gray. Notes on data base operating systems. In *Operating Systems, An Advanced Course*, pages 393–481, London, UK, 1978. Springer-Verlag. ISBN 3-540-08755-9. URL http://dl.acm.org/citation.cfm?id=647433.723863.

[63] Barbara J. Grosz and Sarit Kraus. Collaborative plans for complex group action. *Artificial Intelligence*, 86:269–357, 1996.

[64] Barbara J. Grosz and Candace L. Sidner. Plans for discourse. In P. R. Cohen, J. Morgan, and M. E. Pollack, editors, *Intentions in Communication*, chapter 20, pages 417–444. MIT Press, Cambridge, MA, 1990.

[65] Object Management Group. Data distribution service (dds) specification v1.2. http://www.omg.org/spec/DDS/1.2/ (accessed 2012-05-22), 2007.

[66] Nikolaus Hansen and Andreas Ostermeier. Adapting arbitrary normal mutation distributions in evolution strategies: The covariance matrix adaptation. In *International Conference on Evolutionary Computation*, pages 312–317, 1996.

[67] P. E. Hart, N. J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *Systems Science and Cybernetics, IEEE Transactions on*, 4(2):100–107, 1968. doi:10.1109/TSSC.1968.300136.

[68] W. K. Hartmann. PLANETARY SCIENCE: The Shape of Kleopatra. *Science*, 288:820–821, 2000. doi:10.1126/science.288.5467.820.

[69] Koen Hindriks, Frank S. De Boer, Wiebe Van Der Hoek, and John jules Ch. Meyer. A Formal Embedding of AgentSpeak(L) in 3APL. Technical report, Advanced Topics in Artificial Intelligence, Springer Verlag LNAI 1502, 1998.

[70] Koen V. Hindriks and John-Jules Ch. Meyer. Agent Logics as Program Logics: Grounding KARO. In Christian Freksa, Michael Kohlhase, and Kerstin Schill, editors, *KI*, volume 4314 of *Lecture Notes in Computer Science*, pages 404–418. Springer, 2006.

[71] Koen V. Hindriks, Frank S. De Boer, Wiebe Van Der Hoek, and John-Jules Ch. Meyer. Agent programming in 3APL. *Autonomous Agents and Multi-Agent Systems*, 2(4):357–401, 1999. ISSN 1387-2532.

[72] Steffen Hölldobler and Josef Schneeberger. A new deductive approach to planning. *New Generation Computing*, 8(3):225–244, 1990. doi:10.1007/BF03037518.

[73] Bryan Horling and Victor Lesser. A Survey of Multi-Agent Organizational Paradigms. *The Knowledge Engineering Review*, 19(4):281–316, 2005. URL http://mas.cs.umass.edu/paper/366.

[74] Bryan Horling, Victor Lesser, Regis Vincent, Tom Wagner, Anita Raja, Shelley Zhang, Keith Decker, and Alan Garvey. The TAEMS White Paper, 1999. URL http://mas.cs.umass.edu/paper/182.

[75] Marcus Huber and Edmund H. Durfee. On acting together: Without communication. In *Working Notes of the AAAI Spring Symposium on Representing Mental States and Mechanisms*, pages 60–71, 1995.

[76] *Information technology - XML Metadata Interchange (XMI) – ISO/IEC 19503:2005-11*. International Organization for Standardization, 2005.

[77] Herbert Jaeger and Thomas Christaller. Dual Dynamics: Designing Behavior Systems for Autonomous Robots. *Artificial Life and Robotics*, 2:76–79, 1998.

[78] Nicholas R. Jennings. Controlling cooperative problem solving in industrial multi-agent systems using joint intentions. *Artificial Intelligence*, 75(2): 195–240, 1995.

[79] Richard E. Jones and Rafael Dueire Lins. *Garbage Collection: Algorithms for Automatic Dynamic Memory Management*. John Wiley, 1996. ISBN 0-471-94148-4.

[80] Ari K. Jónsson and Jeremy Frank. A framework for dynamic constraint reasoning using procedural constraints. In Werner Horn, editor, *ECAI 2000, Proceedings of the 14th European Conference on Artificial Intelligence, Berlin, Germany, August 20-25, 2000*, pages 93–97. IOS Press, 2000.

[81] Gal A. Kaminka and Inna Frenkel. Flexible teamwork in behavior-based robots. In Manuela M. Veloso and Subbarao Kambhampati, editors, *Proceedings of The Twentieth National Conference on Artificial Intelligence and the Seventeenth Innovative Applications of Artificial Intelligence Conference, July 9-13, 2005, Pittsburgh, Pennsylvania, USA*, pages 108–113. AAAI Press / The MIT Press, 2005. ISBN 1-57735-236-X.

[82] Gal A. Kaminka and Milind Tambe. Robust agent teams via socially-attentive monitoring. *Journal of Artificial Intelligence Reasearch*, 12:105–147, 2000. doi:10.1613/jair.682.

[83] Hirofumi Katsuno and Alberto O. Mendelzon. On the difference between updating a knowledge base and revising it. In *Proceedings of the Second International Conference on the Principles of Knowledge Representation and Reasoning (KR91)*, pages 387–394. Morgan Kaufmann, 1991.

[84] Kristian Kersting, Martijn Van Otterlo, and Luc De Raedt. Bellman goes relational. In *International Conference on Machine Learning*, pages 465–472. ACM, 2004.

[85] David Kinny, Magnus Ljungberg, Anand S. Rao, Liz Sonenberg, Gil Tidhar, and Eric Werner. Planned team activity. In Cristiano Castelfranchi and Eric Werner, editors, *Artificial Social Systems, 4th European Workshop on Modelling Autonomous Agents in a Multi-Agent World, MAAMAW '92, S. Martino al Cimino, Italy, July 29-31, 1992, Selected Papers*, volume 830 of *Lecture Notes in Computer Science*, pages 227–256. Springer, 1992. ISBN 3-540-58266-5. doi:10.1007/3-540-58266-5_13.

[86] Alexander Kleiner, Christian Dornhege, and Andreas Hertle. RMASBENCH–rescue multi-agent benchmarking. http://kaspar.informatik.uni-freiburg.de/~rslb (accessed 2012-05-12).

[87] C. E. Koksal and H. Balakrishnan. Quality-Aware Routing Metrics for Time-Varying Wireless Mesh Networks. *IEEE Journal on Selected Areas in Communications*, 24(11):1984–1994, November 2006. ISSN 0733-8716. doi:10.1109/JSAC.2006.881637.

[88] Robert A. Kowalski and Marek J. Sergot. A logic-based calculus of events. *New Generation Computing*, 4(1):67–95, January 1986. ISSN 0288-3635. doi:10.1007/BF03037383.

[89] M. R. Krom. The decision problem for a class of first-order formulas in which all disjunctions are binary. *Mathematical Logic Quarterly*, 13(1-2): 15–20, 1967. ISSN 1521-3870. doi:10.1002/malq.19670130104.

[90] H. W. Kuhn. The Hungarian method for the assignment problem. *Naval Research Logistic Quarterly*, 2:83–97, 1955.

[91] Thomas H. Labella, Marco Dorigo, and Jean-Louis Deneubourg. Division of labor in a group of robots inspired by ants' foraging behavior. *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, 1:4–25, September 2006. ISSN 1556-4665. doi:10.1145/1152934.1152936.

[92] Jérôme Lang. Belief update revisited. In *Proceedings of the 20th international joint conference on Artifical intelligence*, pages 2517–2522, San Francisco, CA, USA, 2007. Morgan Kaufmann Publishers Inc.

[93] Nuno Lau, Luis Seabra Lopes, Gustavo Corrente, Nelson Filipe, and Ricardo Sequeira. Robot team coordination using dynamic role and positioning assignment and role based setplays. *Mechatronics*, 21(2):445–454, 2011. ISSN 0957-4158. doi:10.1016/j.mechatronics.2010.05.010. Special Issue on Advances in intelligent robot design for the Robocup Middle Size League.

[94] Kristina Lerman, Chris Jones, Aram Galstyan, and Maja J Mataríc. Analysis of dynamic task allocation in multi-robot systems. *The International Journal of Robotics Research*, 25:225–241, March 2006. ISSN 0278-3649. doi:10.1177/0278364906063426.

[95] V. Lesser, K. Decker, T. Wagner, N. Carver, A. Garvey, B. Horling, D. Neiman, R. Podorozhny, M. Nagendra Prasad, A. Raja, R. Vincent, P. Xuan, and X. Q. Zhang. Evolution of the GPGP/TÆMS Domain-Independent Coordination Framework. *Autonomous Agents and Multi-Agent Systems*, 9(1-2):87–143, July 2004. ISSN 1387-2532. doi:10.1023/B:AGNT.0000019690.28073.04.

[96] Hector J. Levesque. Planning with Loops. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 509–515, 2005.

[97] Hector J. Levesque, Philip R. Cohen, and José H. T. Nunes. On Acting Together. In *Proceedings of AAAI-90*, pages 94–99, Boston, MA, 1990.

[98] Hector J. Levesque, Raymond Reiter, Yves Lespérance, Fangzhen Lin, and Richard B. Scherl. Golog: A logic programming language for dynamic domains. *Journal of Logic Programming*, 31(1–3):59–83, 1997. doi:10.1016/S0743-1066(96)00121-5.

[99] Martin Lötzsch, Max Risler, and Matthias Jüngel. XABSL - A pragmatic approach to behavior engineering. In *Proceedings of IEEE/RSJ International Conference of Intelligent Robots and Systems (IROS)*, pages 5124–5129, Beijing, China, 2006.

[100] John McCarthy and Patrick J. Hayes. Some Philosophical Problems from the Standpoint of Artificial Intelligence. In B. Meltzer and D. Michie, editors, *Machine Intelligence 4*, pages 463–502. Edinburgh University Press, 1969.

[101] D Mills, J Martin, J Burbank, and W Kasch. Rfc 5905 - network time protocol. *Internet Engineering Task Force IETF*, pages 1–111, 2010. URL http://www.rfc-editor.org/info/rfc5905.

[102] Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: Engineering an efficient sat solver. In *Annual ACM IEEE Design and Automation Conference*, pages 530–535. ACM, 2001.

[103] Sreerama K. Murthy. Automatic construction of decision trees from data: A multi-disciplinary survey. *Data Mining and Knowledge Discovery*, 2: 345–389, 1997.

[104] Ranjit Nair, Milind Tambe, and Stacy Marsella. Role allocation and reallocation in multiagent teams: towards a practical analysis. In *Proceedings of the second international joint conference on Autonomous agents and multiagent systems*, AAMAS '03, pages 552–559, New York, NY, USA, 2003. ACM. ISBN 1-58113-683-8. doi:10.1145/860575.860664.

[105] Alexander Nareyek. *Constraint-Based Agents*, volume 2062 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, 2001. ISBN 978-3-540-42258-7. doi:10.1007/3-540-45746-1.

[106] Dana Nau, Okhtay Ilghami, Ugur Kuter, J. William Murdock, Dan Wu, and Fusun Yaman. Shop2: An htn planning system. *Journal of Artificial Intelligence Research*, 20:379–404, 2003.

[107] S. Nema, John Yannis Goulermas, G. Sparrow, and Phil Cook. A hybrid particle swarm branch-and-bound (hpb) optimizer for mixed discrete nonlinear programming. *IEEE Transactions on Systems, Man, and Cybernetics, Part A*, 38(6):1, 2008. doi:10.1109/TSMCA.2008.2003536.

[108] Salam Nema. *Hybrid evolutionary techniques for constrained optimisation design*. PhD thesis, University of Liverpool, 2010.

[109] J. Neyman. On a New Class of Contagious Distributions, Applicable in Entomology and Bacteriology. *The Annals of Mathematival Statistics*, 10: 35–57, 1939.

[110] Trung Thanh Nguyen and Xin Yao. Benchmarking and solving dynamic constrained problems. In *Proceedings of the IEEE Congress on Evolutionary Computation*, CEC'09, pages 690–697, Piscataway, NJ, USA, 2009. IEEE Press. ISBN 978-1-4244-2958-5. doi:10.1109/CEC.2009.4983012.

[111] Shan-Hwei Nienhuys-Cheng and Ronald de Wolf. *Foundations of Inductive Logic Programming*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1997. ISBN 3540629270.

[112] Robert Nieuwenhuis, Albert Oliveras, and Cesare Tinelli. Solving SAT and SAT Modulo Theories: From an abstract Davis–Putnam–Logemann–Loveland procedure to DPLL(T). *J. ACM*, 53:937–977, November 2006. ISSN 0004-5411. doi:10.1145/1217856.1217859.

[113] N. J. Nilsson. *Principles of Artificial Intelligence*. Morgan Kaufmann, San Francisco, CA, USA, 1980.

[114] Thomas Nitsche and Thomas Fuhrmann. A tool for raytracing based radio channel simulation. In *Proceedings of the 4th International ICST Conference on Simulation Tools and Technique*, March 2011.

[115] Hyacinth S. Nwana. Software agents: An overview. *Knowledge Engineering Review*, 11:205–244, 1996.

[116] Object Management Group. OMG Unified Modeling Language™ (OMG UML), Superstructure, May 2010. URL http://www.omg.org/spec/UML/2.3/Superstructure/PDF/. OMG Document Number: formal/2010-05-05.

[117] Boon Hua Ooi and Aditya K. Ghose. Constraint-based agent specification for a multi-agent stock brokering system. In *Proceedings of the 12th international conference on Industrial and engineering applications of artificial intelligence and expert systems: multiple approaches to intelligent systems*, IEA/AIE '99, pages 409–419, Secaucus, NJ, USA, 1999. Springer-Verlag New York, Inc. ISBN 3-540-66076-3. URL http://dl.acm.org/citation.cfm?id=341506.341611.

[118] Lynne E. Parker. Alliance: An architecture for fault tolerant multi-robot cooperation. *IEEE Transactions on Robotics and Automation*, 14:220–240, 1998.

[119] David Payton, Mike Daily, Regina Estowski, Mike Howard, and Craig Lee. Pheromone robotics. *Autonomous Robots*, 11:319–324, 2001. ISSN 0929-5593. doi:10.1023/A:1012411712038.

[120] Judea Pearl. *Heuristics – intelligent search strategies for computer problem solving*. Addison-Wesley series in artificial intelligence. Addison-Wesley, 1984. ISBN 978-0-201-05594-8.

[121] Adrian Petcu. *A Class of Algorithms for Distributed Constraint Optimization*. Phd. thesis no. 3942, Swiss Federal Institute of Technology (EPFL), Lausanne (Switzerland), October 2007. URL http://liawww.epfl.ch/Publications/Archive/Petcu2007thesis.pdf.

[122] G. D. Plotkin. A Structural Approach to Operational Semantics. Technical Report DAIMI FN-19, University of Aarhus, University of Aarhus, 1981.

[123] Armand E. Prieditis. Machine discovery of effective admissible heuristics. In *Machine Learning*, pages 117–141, 1993.

[124] Patrick Prosser. Hybrid Algorithms for the Constraint Satisfaction Problem. *Computational Intelligence*, 9:268–299, 1993.

[125] D. Pynadath and M. Tambe. Multiagent teamwork: Analyzing the optimality and complexity of key theories and models. In *Proceedings of the 1st conference of autonomous agents and multiagent systems (AAMAS-2002)*, 2002. URL http://citeseer.ist.psu.edu/article/pynadath02multiagent.html.

[126] David V. Pynadath, Milind Tambe, and Nicolas Chauvat. Toward team-oriented programming. In *Intelligent Agents VI: Agent Theories, Architectures, and Languages*, pages 233–247, 1999.

[127] Morgan Quigley, Ken Conley, Brian P. Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y. Ng. Ros: an open-source robot operating system. In *ICRA Workshop on Open Source Software*, 2009.

[128] J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1:81–106, March 1986. ISSN 0885-6125. doi:10.1023/A:1022643204877.

[129] Anand S. Rao. AgentSpeak(L): BDI agents speak out in a logical computable language. In *MAAMAW '96: Proceedings of the 7th European workshop on Modelling autonomous agents in a multi-agent world : agents breaking away*, pages 42–55, Secaucus, NJ, USA, 1996. Springer-Verlag New York, Inc. ISBN 3-540-60852-4.

[130] Anand S. Rao, Michael P. Georgeff, and E. A. Sonenberg. Social plans: A preliminary report. In E. Werner and Y. Demazeau, editors, *Decentralized AI 3 — Proceedings of the Third European Workshop on Modelling Autonomous Agents in a Multi-Agent World (MAAMAW-91)*, pages 57–76, Kaiserslautern, Germany, 1992. Elsevier Science B.V.: Amsterdam, Netherland. URL http://citeseer.ist.psu.edu/rao92social.html.

[131] Roland Reichle. *Information Exchange and Fusion in Heterogeneous Distributed Environments*. Phd thesis, University of Kassel, Kassel, 2010.

[132] Luís Paulo Reis, Nuno Lau, and Eugenio Oliveira. Situation based strategic positioning for coordinating a team of homogeneous agents. In *Balancing Reactivity and Social Deliberation in Multi-Agent Systems, From RoboCup to Real-World Applications (selected papers from the ECAI 2000 Workshop and additional contributions)*, pages 175–197, London, UK, 2001. Springer-Verlag. ISBN 3-540-42327-3. URL http://dl.acm.org/citation.cfm?id=646142.681100.

[133] Raymond Reiter. The frame problem in the situation calculus: A simple solution (sometimes) and a completeness result for goal regression. In Vladimir Lifschitz, editor, *Artificial Intelligence and Mathematical Theory of Computation: Papers in Honor of John McCarthy*, pages 359–380. Academic Press, San Diego, CA, 1991.

[134] Raymond Reiter. *Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems*. The MIT Press, Massachusetts, MA, 2001. ISBN 0262182181.

[135] David Richardson. Some Unsolvable Problems Involving Elementary Functions of a Real Variable. *Journal of Symbolic Logic*, 33:5114–520, 1968.

[136] Martin Riedmiller and Heinrich Braun. Rprop - a fast adaptive learning algorithm. *International Symposium on Computer and Information Sciences-ISCIS*, 1992.

[137] RoboCup. RoboCup Foundation. http://www.robocup.org/ (accessed 2012-04-10).

[138] Francesca Rossi, Peter van Beek, and Toby Walsh. *Handbook of Constraint Programming (Foundations of Artificial Intelligence)*. Elsevier Science Inc., New York, NY, USA, 2006. ISBN 0444527265.

[139] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Pearson Eduction International, 2nd edition, 2003. ISBN 0-13-080302-2.

[140] Earl D. Sacerdoti. A structure for plans and behavior. Technical Report 109, AI Center, SRI International, 333 Ravenswood Ave., Menlo Park, CA 94025, August 1975.

[141] Sebastian Sardina and Lin Padgham. Goals in the context of BDI plan failure and planning. In *Proceedings of the 6th International Joint Conference on Autonomous Agents and Multiagent systems*, AAMAS '07, pages 7:1–7:8, New York, NY, USA, 2007. ACM. ISBN 978-81-904262-7-5. doi:10.1145/1329125.1329134.

[142] Paul Scerri, David V. Pynadath, Nathan Schurr, Alessandro Farinelli, Sudeep Gandhe, and Milind Tambe. Team oriented programming and proxy agents: The next generation. In Mehdi Dastani, Juergen Dix, and Amal El Fallah-Seghrouchni, editors, *PROMAS*, volume 3067 of *Lecture Notes in Computer Science*, pages 131–148. Springer, 2003. ISBN 3-540-22180-8.

[143] Thomas J. Schaefer. The complexity of satisfiability problems. In *Proceedings of the tenth annual ACM Symposium on Theory of Computing*, STOC '78, pages 216–226, New York, NY, USA, 1978. ACM. doi:10.1145/800133.804350.

[144] Andreas Scharf. Grafische Verhaltensmodellierung kooperativer autonomer Robotersysteme. Bachelor thesis, University of Kassel, Germany, 2008.

[145] Arne Schmitz and Leif Kobbelt. Wave propagation using the photon path

map. In *Proceedings of the 3rd ACM international workshop on Performance evaluation of wireless ad hoc, sensor and ubiquitous networks*, PE-WASUN '06, pages 158–161, New York, NY, USA, 2006. ACM. ISBN 1-59593-487-1. doi:10.1145/1163610.1163638.

[146] J. Schubert and H. Sidenbladh. Sequential clustering with particle filters – estimating the number of clusters from data. In *8th International Conference on Information Fusion*, volume 1, pages 1–8. ISIF, July 2005. doi:10.1109/ICIF.2005.1591845.

[147] Bart Selman, Henry A. Kautz, and Bram Cohen. Noise strategies for improving local search. In *Proceedings of the Eleventh National Conference on Artificial Intelligence (AAAI-94)*, pages 337–343, 1994.

[148] M. Senel, K. Chintalapudi, Dhananjay Lal, A. Keshavarzian, and E. J. Coyle. A Kalman Filter Based Link Quality Estimation Scheme for Wireless Sensor Networks. In *Global Telecommunications Conference, 2007. GLOBECOM '07. IEEE*, pages 875–880. IEEE, November 2007. ISBN 978-1-4244-1043-9. doi:10.1109/GLOCOM.2007.169.

[149] Glenn Shafer. *A Mathematical Theory of Evidence*. Princeton University Press, Princeton, 1976.

[150] Murray Shanahan. *Solving the Frame Problem: A Mathematical Investigation of the Common Sense Law of Inertia*. MIT Press, 1997.

[151] Murray Shanahan. The event calculus explained. In *Artificial Intelligence Today: Recent Trends and Developments*, volume 1600 of *Lecture Notes in Computer Science*, pages 409–430. Springer, 1999.

[152] Yi Shang, Markus P.J. Fromherz, and Lara Crawford. A new constraint test-case generator and the importance of hybrid optimizers. *European Journal of Operational Research*, 173(2):419–443, September 2006.

[153] Alex Shtof. Autodiff – high-performance and high-accuracy automatic function-differentiation library suitable for optimization and numeric computing. http://autodiff.codeplex.com/ (accessed 2012-05-21).

[154] J.C. Simon and O. Dubois. Number of solutions to satisfiability instances – applications to knowledge base. *International Journal of Pattern Recognition and Artificial Intelligence*, 3:53–65, 1989.

[155] Hendrik Skubch. Solving non-linear arithmetic constraints in soft realtime environments. In *27th Symposium On Applied Computing*, volume 1, pages

67–75. ACM, 2012.

[156] Hendrik Skubch and Michael Thielscher. Strategy learning for reasoning agents. In João Gama, Rui Camacho, Pavel Brazdil, Alípio Jorge, and Luís Torgo, editors, *Machine Learning: ECML 2005, 16th European Conference on Machine Learning, Porto, Portugal, October 3-7, 2005, Proceedings*, volume 3720 of *Lecture Notes in Computer Science*, pages 733–740. Springer, 2005. ISBN 3-540-29243-8. doi:10.1007/11564096_75.

[157] Hendrik Skubch, Michael Wagner, Roland Reichle, Stefan Triller, and Kurt Geihs. Towards a comprehensive teamwork model for highly dynamic domains. In Joaquim Filipe, Ana L. N. Fred, and Bernadette Sharp, editors, *ICAART 2010 - Proceedings of the International Conference on Agents and Artificial Intelligence, Volume 2 - Agents, Valencia, Spain, January 22-24, 2010*, pages 121–127. INSTICC Press, 2010. ISBN 978-989-674-022-1.

[158] Hendrik Skubch, Daniel Saur, and Kurt Geihs. Resolving conflicts in highly reactive teams. In Norbert Luttenberger and Hagen Peters, editors, *17th GI/ITG Conference on Communication in Distributed Systems, KiVS*, volume 17 of *OASICS*, pages 170–175. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany, 2011. ISBN 978-3-939897-27-9. doi:10.4230/OASIcs.KiVS.2011.170.

[159] Hendrik Skubch, Michael Wagner, Roland Reichle, and Kurt Geihs. A modelling language for cooperative plans in highly dynamic domains. *Mechatronics*, 21(2):423–433, 2011. Special Issue on Advances in intelligent robot design for the Robocup Middle Size League.

[160] Jon Sneyers, Tom Schrijvers, and Bart Demoen. The computational power and complexity of constraint handling rules. *ACM Trans. Program. Lang. Syst.*, 31(2):8:1–8:42, February 2009. doi:10.1145/1462166.1462169.

[161] David Steinberg, Frank Budinsky, Marcelo Paternostro, and Ed Merks. *EMF: Eclipse Modeling Framework 2.0*. Addison-Wesley Professional, 2009. ISBN 0321331885.

[162] Jayoung Sung, Henrik I. Christensen, and Rebecca E. Grinter. Sketching the Future: Assessing User Needs for Domestic Robots. In *The 18th IEEE International Symposium on Robot and Human Interactive Communication*, pages 153–158, 2009. doi:10.1109/ROMAN.2009.5326289.

[163] Milind Tambe. Towards flexible teamwork. *Journal of Artificial Intelligence Research*, 7:83–124, 1997. URL http://citeseer.ist.psu.edu/tambe97towards.html.

[164] Milind Tambe, David V. Pynadath, and Nicolas Chauvat. Building dynamic

agent organizations in cyberspace. *IEEE Internet Computing*, 4:65–73, March 2000. ISSN 1089-7801. doi:10.1109/4236.832948.

[165] Russell Taylor and Leo Joskowicz. Computer-integrated surgery and medical robotics. *The Hand*, 2002:1199–1222, 2008.

[166] Michael Thielscher. Introduction to the fluent calculus. *Electronic Transactions on Artificial Intelligence*, 2:179–192, 1998. URL http://www.ep.liu.se/ej/etai/1998/006/.

[167] Michael Thielscher. Flux: A logic programming method for reasoning agents. *Theory and Practice of Logic Programming*, 5:533–565, July 2005. ISSN 1471-0684. doi:10.1017/S1471068405002358.

[168] Michael Thielscher. *Reasoning robots: the art and science of programming robotic agents*. Applied logic series. Springer, 2005.

[169] Marc Toussaint, Nils Plath, Tobias Lang, and Nikolay Jetchev. Integrated motor control, planning, grasping and high-level reasoning in a blocks world using probabilistic inference. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2010.

[170] Douglas Vail and Manuela Veloso. Dynamic multi-robot coordination. In *Multi-Robot Systems: From Swarms to Intelligent Automata, Volume II*, pages 87–100. Kluwer Academic Publishers, 2003.

[171] Pascal Van Hentenryck. Numerica: a modeling language for global optimization. In *Proceedings of the Fifteenth international joint conference on Artifical intelligence - Volume 2*, pages 1642–1647, San Francisco, CA, USA, 1997. Morgan Kaufmann Publishers Inc. ISBN 1-555860-480-4. URL http://dl.acm.org/citation.cfm?id=1622270.1622392.

[172] Pascal Van Hentenryck and Thierry Le Provost. Incremental search in constraint logic programming. *New Generation Computing*, 9:257–275, 1991. ISSN 0288-3635. doi:10.1007/BF03037165.

[173] Gérard Verfaillie and Thomas Schiex. Solution reuse in dynamic constraint satisfaction problems. In *Proceedings of the twelfth national conference on Artificial intelligence (vol. 1)*, AAAI '94, pages 307–312, Menlo Park, CA, USA, 1994. American Association for Artificial Intelligence. ISBN 0-262-61102-3. URL http://dl.acm.org/citation.cfm?id=199288.178066.

[174] Felix von Leitner. The Dark Side of C++. http://www.fefe.de/c++/c%2B%2B-talk.pdf (accessed 2012-05-22), 2007. Presented at Chaos Communication Camp, Berlin, 2007.

[175] Tingting Wang, Jiming Liu, and Xiaolong Jin. Minority game strategies

in dynamic multi-agent role assignment. In *Proc. of the IEEE/WIC/ACM International Conference on Intelligent Agent Technology*, pages 316–322, Washington, DC, USA, 2004. IEEE Computer Society. ISBN 0-7695-2101-0. doi:10.1109/IAT.2004.79.

[176] J. H. Ward. Hierarchical Grouping to Optimize an Objective Function. *Journal of the American Statistical Association*, 58(301):236–244, March 1963. ISSN 01621459. doi:10.2307/2282967.

[177] Jörg Weber and Franz Wotawa. Combining runtime diagnosis and ai-planning in a mobile autonomous robot to achieve a graceful degradation after software failures. In Joaquim Filipe, Ana L. N. Fred, and Bernadette Sharp, editors, *ICAART 2010 - Proceedings of the International Conference on Agents and Artificial Intelligence, Volume 1 - Artificial Intelligence, Valencia, Spain, January 22-24, 2010*, pages 127–134. INSTICC Press, 2010. ISBN 978-989-674-021-4.

[178] T. Weigel, J.-S. Gutmann, M. Dietl, A. Kleiner, and B. Nebel. CS Freiburg: Coordinating robots for successful soccer playing. *IEEE Transactions on Robotics and Automation*, 18(5):685–699, 2002.

[179] Terry Winograd. *Understanding Natural Language*. Academic Press, New York, 1972. doi:10.1002/bs.3830180608.

[180] D. H. Wolpert and W. G. Macready. No free lunch theorems for optimization. *Evolutionary Computation, IEEE Transactions on*, 1(1):67–82, April 1997. ISSN 1089-778X. doi:10.1109/4235.585893.

[181] Michael Wooldridge. *An Introduction to MultiAgent Systems*. Wiley Publishing, 2nd edition, 2009. ISBN 0470519460, 9780470519462.

[182] Michael Wooldridge, Nicholas Jennings, and David Kinny. A methodology for agent-oriented analysis and design. *Journal of Autonomous Agents and Multi-Agent Systems*, 3:285–312, 1999.

[183] John Yen, Jianwen Yin, Thomas R. Ioerger, Michael S. Miller, Dianxiang Xu, and Richard A. Volz. Cast: Collaborative agents for simulating teamwork. In *Proceedings of the 17th International Joint Conference on Artificial intelligence - Volume 2*, IJCAI'01, pages 1135–1142, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc. ISBN 1-55860-812-5, 978-1-558-60812-2.

[184] Jianwen Yin, Michael S. Miller, Thomas R. Ioerger, John Yen, and Richard A. Volz. A knowledge-based approach for designing intelligent team training systems. In Carles Sierra, Maria Gini, and Jeffrey S. Rosenschein, editors, *Proceedings of the Fourth International Conference on Autonomous Agents*, pages 427–434, Barcelona, Catalonia, Spain, 2000. ACM

Press. doi:10.1145/336595.337560.

[185] Weixiong Zhang, Guandong Wang, Zhao Xing, and Lars Wittenburg. Distributed stochastic search and distributed breakout: properties, comparison and applications to constraint optimization problems in sensor networks. *Artificial Intelligence*, 161(1-2):55–87, January 2005. ISSN 0004-3702. doi:10.1016/j.artint.2004.10.004.

[186] H.-J. Zimmermann. *Fuzzy set theory—and its applications (3rd ed.)*. Kluwer Academic Publishers, Norwell, MA, USA, 1996. ISBN 0-7923-9624-3.

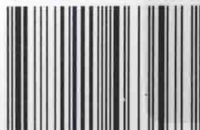
[187] Oliver Zweigle, Reinhard Lafrenz, Thorsten Buchheim, Uwe-Philipp Käppeler, Hamid Rajaie, Frank Schreiber, and Paul Levi. Cooperative agent behavior based on special interaction nets. In Tamio Arai, Rolf Pfeifer, Tucker R. Balch, and Hiroshi Yokoi, editors, *Proceedings of the 9th International Conference on Intelligent Autonomous Systems - IAS, University of Tokyo, Tokyo, Japan, March 7-9, 2006*, pages 651–659. IOS Press, 2006. ISBN 1-58603-595-9.

国际视野　科技前沿

国际电气工程先进技术译丛

传播国际最新技术成果　搭建电气工程技术平台

《自主移动机器人行为建模与控制》
《覆冰与污秽绝缘子》
《高压直流输电——功率变换在电力系统中的应用》
《MATLAB数值分析方法在电气工程中的应用》
《太阳能利用技术及工程应用》
《超级电容器的应用》
《小型风力机：分析、设计与应用》
《太阳电池、LED和二极管的原理：PN结的作用》
《风力发电系统 —— 技术与趋势》
《可持续电力系统的建模与控制：面向更为智能和绿色的电网》
《电力系统高级预测技术和发电优化调度》
《大规模储能技术》
《风电系统电能质量和稳定性对策》
《环境能源发电：太阳能、风能和海洋能》
《传热学：电力电子器件热管理》
《现代电力电子学与交流传动》
《功率半导体器件：原理、特性和可靠性》
《风能系统——实现安全可靠运行的优化设计与建设》
《储能技术》
《光伏系统工程》（原书第3版）
《光伏与风力发电系统并网变换器》
《车辆能量管理：建模、控制与优化》
《纯电动及混合动力汽车设计基础》（原书第2版）
《电动汽车技术、政策与市场》
《永磁无刷电机及其驱动技术》
《先进电气驱动的分析、建摸与控制》
《智能电网可再生能源系统设计》
《风力发电工程指南》
《用于制造固体氧化物燃料电池的钙钛矿型氧化物》
《太阳能物理》
《柔性交流输电系统在电网中的建模与仿真》
《风电并网：联网与系统运行》
《可再生能源的转换、传输和储存》
《海底电力电缆——设计、安装、修复和环境影响》
《光伏技术与工程手册》
《风力发电的模拟与控制》
《风电场并网稳定性技术》
《智能电网中的电力电子技术》
《电磁屏蔽原理与应用》
《高效可再生分布式发电系统》
《电网保护》
《分布式发电——感应和永磁发电机》
《电力系统谐波》
《风能与太阳能发电系统——设计、分析与运行》（原书第2版）
《瞬时功率理论及其在电力调节中的应用》
《风力机控制系统原理、建模及增益调度设计》
《高压输配电设备实用手册》
《电力变流器电路》
《电力系统中的电磁兼容》
《超高压交流输电工程》（原书第3版）
《高压直流输电与柔性交流输电控制装置——静止换流器在电力系统中的应用》
《配电可靠性与电能质量》



上架指导　工业技术／电子技术／机器人

ISBN 978-7-111-46357-3

定价：59.90元